COMMODORE MAGAZINE

VOL 2
ISSUE 6

★ **Keyboard Coding**
★ **Printer Formatting**
★ **Hard Disk Drives**

Registered by Australia Post
Publication No. NBP4288

REGISTERED FOR POSTING AS A PUBLICATION: CATEGORY B

The objective of this magazine is to disseminate information to all users of Commodore computer products. This magazine contains a variety of information collected from other Commodore publications, and generated locally. Contributions from all Commodore User Groups, and individual users are encouraged.

Advertising space is restricted and allocated on a first-come, first-served basis. Advertising rates can be supplied on request.

All copy and advertising should be addressed to:

The Editor,
COMMODORE MAGAZINE,
P.O. BOX 336, ARTARMON
N.S.W. 2064
AUSTRALIA

ISSUE No.	COPY/ADV DEADLINE	PUBLICATION DATE
1	February 18th	March 6th
2	April 1st	April 17th
3	May 13th	May 29th
4	June 17th	July 3rd
5	July 29th	August 14th
6	September 9th	September 25th
7	October 14th	October 30th
8	November 25th	December 4th

Production & Typesetting: Mervyn Beamish Graphics Pty. Ltd. 82 Alexander Street, Crows Nest 2065 Phone 439 1827

Printer: Liberty Print 108 Chandos Street, Crows Nest 2067 Phone 43 4398

SUBSCRIPTIONS

	Annual Subscription	Single Copy
Postage paid within Australia	$A30.00	$A5.00
Overseas Postage Paid	$A38.00	$A6.00

Subscriptions to COMMODORE MAGAZINE can be obtained from your authorised Commodore Dealer, or from:

COMMODORE MAGAZINE,
P.O. BOX 336, ARTARMON,
N.S.W. 2064,
AUSTRALIA.

Vol 1 1981
Vol 2 1982

Typeset and assembled off Commodore Wordcraft disks

PLEASE NOTE: To provide a good information service to Commodore users, we will regularly mention equipment, software and services offered by companies and individuals not directly related to Commodore. In doing so, we are not making recommendations, and cannot be responsible for the validity and accuracy of any statements made.

EDITOR'S DESK . . .

The subscriptions to the Commodore Magazine show a large interest from VIC-20 users and we try to balance the content of the magazine to suit all readers.

However from time to time we receive comments that the magazine does not have articles of interest to a particular reader. What we would like to know, is what you would like to see. Please drop us a line on the content you would like to see in the magazine, to advance your knowledge of the Commodore product range.

We aim to cover topics that are of use to beginners and experienced Commodore users and sometimes articles may appear that are general knowledge to Commodore users but may be the starting key to a newcommer to this fascinating world of computing.

Communications are rapidly becomming a facit of personal computing. We now have low priced modems that allow you to simply connect your computer to the telephone network to communicate to the vast world of data banks, information networks and other computer users.

Commodore is supporting the Australian Beginning and we are working on setting up a bulletin board that will provide up to the minute product information, programming advice, service details and of course a mail facility directly into Commodore.

The aim is to provide Public Domain software for the complete range of Commodore computers to all subscribers and generate more interest in the electronic mail concept.

table of contents

next issue:

VIC Special
—Update On The VIC-20 Product Line Including Hardware And Software Available
SUPERPET Update

list of advertisers

Commodore Announces 'Easy' Spreadsheet Programs for SuperPET and COMMODORE 64 Microcomputers

A "family" of four "Easy" spreadsheet programs, designed to provide Commodore users with lightning-fast decision-making tools to be used in a business environment, has been introduced by Commodore Business Machines.

The four new programs, EasyCalc, EasyPlot, EasyFinance and EasyScan, will be available in Australia very soon, and will initally run on SuperPET and Commodore 64 micros. In the near future, the "Easy" programs will also be available for other Commodore micros including the recently announced "P" and "B" series.

EasyCalc is the biggest micro worksheet available with 65 columns and 999 rows. It replaces old fashioned paper, pen and calculator and saves many hours in solving a wide range of numerical problems such as budgets, cash flow forecasting, product and resource planning, stock fluctuations, engineering or scientific data, trend analysis, agricultural yields and floor plan models.

EasyCalc has "help" functions to aid first-time users and experts alike. It has the ability to print out all formulas and assumptions, and its disk-based grid allows automatic matrix consolidations. Additional features include movement of data in a matrix, selective row reporting and printing, instant "what if" calculations, and integration with Commodore's EasyPlot spreadsheet program.

Commodore's EasyPlot allows users to intepret and present numerical data with high quality charts and graphs, and produces bar and pie charts, scatter diagrams and line graphs. It analyzes relationships between different data sets, and charts population trends and stock price fluctuations as well as othe numerical information. The user-oriented format of EasyPlot features full-page printing of all charts and graphs and integration with EasyCalc.

EasyFinance from Commodore allows users to analyze alternatives and see pros and cons so vital in decision-making. Its features include lease analysis, full loan amortization functions, present and future cash flow calculations and discounted cash flow analysis to help make decisions regarding borrowing, tax deductions and the cost of inflation.

The fourth member of the new spreadsheet family from Commodore, EasyScan, is an instant access diary that never lets you forget. It features day, week, month or year at-a-glance, organization of data by priorities and "zoom" for in-depth scan. EasyScan allows users to schedule time and resources for optimum productivity and also serves as an appointment secretary and project planner/task coordinator.

Announcements of the availability and pricing for these products will be made in the Commodore Magazine as soon as they come to hand.

DIRECT CONNECT MODEM

A Telecom Australia approved direct connect modem that can be used on the VIC-20 computer, or on any suitably interfaced Commodore computer, has been released by Dick Smith.

This low cost modem is a must for any VIC owner, allowing access to the many data bases that already exist, or to exchange programs and data between other computer owners via the telephone line.

A review of the product is on page 6 of this issue.

VIC PRODUCTS ANNOUNCED

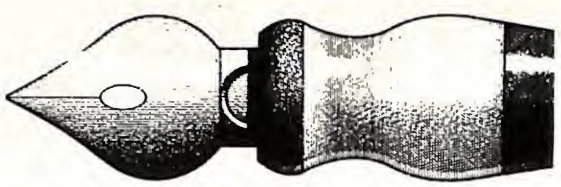
C.W. Electronics from Quuensland have announced a range of VIC products and accessories.

These include;

- 40/80 column expansion
- VIC telephone dialler
- VIC EPROM programmer
- VIC Wordcraft
- RTTY Interfaces

Reviews of many of these products will be included in future issues of the Commodore Magazine. Refer to the advertisement on page 18 for further information.

LETTERS TO THE EDITOR



Dear Editor.

I have just purchased a CBM 4016, know nothing about programming, but am having a lot of fun putting in the programs that I found in back issues of the Commodore Magazine.

It appears that Commodore has a number of different computers, PET, 2.0, 4.0, VIC-20 and others – would you please advise if the term PET is used for most Commodore computers or strictly restricted to PET computers. I have noticed a vast reference in the magazine to 'your PET' and wonder if the remarks apply to my machine.

Also, will programs published for the VIC-20 run on my machine? It would be helpful in program articles if it could be clearly stated on which computers they apply, mainly for beginners like me.

PETER BUNGAY.

Thank you for your letter. We will endevour to make sure that the programs are qualified as to the type of computer they will run on.

Most VIC-20 programs will run on the other Commodore computers if they are programmed in BASIC only and reference to the VIC features of colour and sound are omitted. You do have to watch for references to PEEK and POKE commands, as the locations referenced may change from one computer to another.

The PET computer was the first computer manufactured by Commodore.

It had a calculator styled keyboard and 8K of user memory.

Eventually this was replaced with a graphics keyboard version, still 40 column and was called the 3000 series in Australia and the 2000 series in USA. The two models were the PET 3016 and PET 3032 which refered to the user memory capacity. The 3000 series had BASIC version 2.0. (In fact the same BASIC as used in the VIC-20.)

To support a new version of the Disk Operating System, Commodore released the 4000 series which had two models, the 4016 and 4032. These had BASIC version 4.0 as standard. An upgrade is available for the 3000 series to become identical to the 4000 series.

The term PET has been loosely maintained with the 40 column computers and the term CBM has been used with the 80 column computers, however the word PET is synonomous with Commodore and many people use it when referring to any of the Commodore computers.

To compound the confusion, the 9000 series has been titled the SuperPET, even though it has an 80 column screen.

Dear Editor.

I have recently obtained a CBM 8032 computer and am interested in learning more about assemblers and compilers. Are there any products available to assist me.

PETER MACKAY.

Commodore markets the DTL BASIC Compiler in Australia and we feel that this is most suitable for increasing the execution speed of a BASIC program, reducing the size of the program, and providing protection to the source coding.

An Assembler Development Program is available as Public Domain Software, provided by Commodore.

Both these products are available from your Commodore Dealer.

VIC REVEALED

Steven Hall

Now that a six month period has passed since the introduction of the VIC-20, lets look back at the achievements and failings of the product.

Technology is advancing at an enormous rate, and already Commodore have announced a new range of equipment to compliment the popular VIC series. It is, in fact, a sign of the times when insurance underwriters will write-off computer assets in only 3-5 years – due to technological advances.

There are few other fields where the general public is able to buy products only months away from its original design stage, so in a way, the end-users are also riding the technological wave of computer advancement. Which brings in probably the major criticism of our industry, software development is unable to keep up with these advances. One can buy a computer just released on the market and find little software available to run on it. Unfortunately, the end-user wants the latest hardware with the software support of a product that has been two years on the market.

The problem is one of education in the home computer field, you cannot have the latest technology together with a wealth of established software. The business world has learnt from this lesson and you are now finding the established companies and products making the greatest gains, whereas the new computer companies on the market are fighting fiercely for a smaller portion of the pie.

With the VIC-20, I feel the Australian market has been extremely lucky. With only 6 months gone, one can open any of the popular computer magazines and find dozens of enterprising companies importing recently developed software from the European and American markets.

Some people thought the Australian market was at a disadvantage having a release date 6-12 months after the mass markets of America and Europe. But now we can pick and choose whose software we want, and still have the latest machines on the market.

Also, here in Australia there is a very real grass roots industry developing with companies from all over the continent saying "Hey, we need Australian software, lets develop our own", and are now releasing it on the local markets. Programs like data capture systems, word processing, tutorials in subjects such as languages and maths are being released almost daily.

We now have a networking system for VIC-20's in schools, which mean that our own children are being exposed to computers from primary level upwards.

Another advantage of being on the forefront of computer development is the fact that the cost is going down, not up – as is 99% of all high technological fields. You can have a 21,000 byte computer for only $528, you would have had to pay more than double that, in real money terms, only two years ago. Accessories like disk drives are plummeting in price, with the aid of further discoveries in storage medias.

So, where are we headed? We have seen the start of the universal computer with machines like the new Commodore 64, which if rumors are true, will eventually be able to emulate other computer languages. As to whether it will be in this new generation of computers or the next, remains to be seen.

Computer awareness and education has started to boom – already in the last three months I personally know of four full time VIC-20 computer schools being established in the Sydney area alone! But the biggest impact is in the home, for both educational and entertainment. It is here that the most significant change has taken place. With a growth that is almost breathtaking, parents have shown a computer awareness and are purchasing home computers to give their children first-hand experience of computer usage and using these machines to further their families education.

I, for one, look forward to that generation that will develop the home computer to be mankinds greatest tool since the wheel.

WE WROTE THE BOOK ON SOFTWARE!

OVER 1,000 APPLICATIONS ONLY $14.95

The Commodore Software Encyclopedia is a comprehensive directory of over 1000 programs for business, education, personal, and recreational applications. Listings are provided for such categories as:

- Business
  including Dow Jones Portfolio Management, Legal Time Accounting, Accounts Receivable, Inventory Systems, Tax Preparation, Real Estate Programs, Payroll, General Accounting, Mailing List Management, and Visicalc.
- Word Processing
- Utilities
- Engineering Aids
- Personal Aids
- Games
- VIC-20
- Publications
- Software Vendors
- Hardware Overview
- Overseas Suppliers
- Education
  including Math, Science, Social Studies, Verbal Skills, and Administration.

Whether you're interested in personal computers or own a Commodore, this directory can be an invaluable one in locating just the right software package for you.

Pick one up at your local Commodore dealer today or return this coupon.

Note the directory lists programs available mainly from North American suppliers. Agents for many of these products are in Australia.

Please send. . . . . .Software Encyclopedia(s) $14.95 + $3.00 postage and handling. Enclose cheque or money order.

Name _____

Address _____

_____

_____

_____

Return to: Computer Reference Guide
204/254 Victoria Ave.
Chatswood 2067

commodore COMPUTER

Product Review

DIRECT CONNECT MODEM FOR VIC & CBM

For all those enthusiasts and remote business users who find there computing needs can be further fulfilled by having the ability to communicate with other users and systems then, until now the only way was to use an accoustically coupled modem.

This somewhat awkward means of data communication necessitated that the user had to lift his telephone receiver, dial the second parties number and wait for a connection, at this point the phone receiver then had to be placed into the rubber cups of the modem. A rather messy operation let alone worry about the computer.

Now all of that is old hat, since Dick Smith have released their 300 baud, full duplex direct connect modem.

The unit just plugs straight into the original telecom installed socket. The input to the modem is RS232-C standard so for your VIC-20 you will require a VIC 1011A interface cartridge and on CBM machines a suitable IEEE-RS232 interface.

To use the unit the computer must be loaded with some form of communication software. The modem set to "phone" position on the "phone – modem" select switch and the appropriate position on the "answer – originate" switch.

To establish communication the phone receiver is lifted and the number dialed, on hearing the receiving parties modem output tone, the "phone – modem" switch can then be set the "modem" position, the green LED will then light to indicate carrier detect.

The phone receiver can then be hung-up. On completion of operation the "phone – modem" switch must be returned to the phone position.

The VIC-20 was tested using the "VICterm" program from the "COMMS Terminal" package (Code 310135). Successful communication was established with the "Australian Beginning" and other data bases.

The only limitations on the unit were:-

1. The need for telecom installation
2. When used with low signal phones i.e. PABX systems results tended to be unreliable.

Apart from these, the advantages far outway the disadvantages, the best being the price, $169.00.

Further details can be obtained from your local Dick Smith retailer.

VARIABLE MANIPULATION

by Brad Fisher

I dabble in machine code a bit, and I've written a few stand-alone routines that work by themselves, but I've always been terrified of writing a routine that must interact with BASIC. Then one day, the situation arose where I had to modify some variables in machine code, that had been created by BASIC. So daunted was I that I almost abandoned the program! But, what the heck, I could at least give it a go.

I saw how relatively simple it was (when compared to quantum physics), but still wasn't sure. I played around a bit, just to get the feel of it, and wrote some short routines.

One routine swapped two variables. It seemed such a simple thing to do. (Skip this part if you aren't interested in the guts of it.)

All simple variables are stored in a tables at the end of your BASIC program. Each entry in the table consists of 7 memory locations of BYTES.

NAME 1	NAME 2	VALUE	VALUE	0	0	0	Integer Variables
1	2	3	4	5	6	7	
NAME 1	NAME 2	VALUE	VALUE	VALUE	VALUE	VALUE	Float. Point Variables
1	2	3	4	5	6	7	
NAME 1	NAME 2	LENGTH	POINTER	POINTER	0	0	String Variables

The first two bytes in each type of variable contains the name of the variable. Incidentally, that is the reason why only the first two characters of a variable are recognised – BALL$ and BACK$ are the same variable – BA$.

These two characters also contain coded information as to what type of variable it is. For example, if the variable is AA, then bytes one and two would contain 65 and 65 (CHR$(65)='A').

If both characters are less than 128, it is a floating point variable. If both are greater than 127, it is an integer variable, e.g. AA% would fill bytes 1 and 2 with 65+128 and 65+128.

A string (AA$) would produce 65 and 65+128 in the first two bytes.

In floating point variables, the next 5 bytes contain the actual value of the variable, encoded in 'Binary Coded Decimal'. (Don't ask me what it means – I'm not too sure myself).

Integer variables contain the actual value in the next two bytes, labelled 3 and 4. Multiply the contents of byte 3 by 256 and add bytes 4, and you have the value of the integer variable, sort of. If the byte 3 is greater than 127, then the integer value is negative. Bytes 5, 6, and 7 contain zeros, to fill out to seven bytes. (As you can see, no space is saved by using integer variables instead of floating point variables).

String variables do not contain the characters they represent. Instead, the third byte contains the length of the string. (This is the reason why a string cannot exceed 255 characters. One byte can only contain a number in the range 0-255.) The 4th and 5th bytes contain a pointer to where the string can be found. If the string is a literal, for instance B$='HELLO' then the pointer will point to the letter 'H' in the BASIC text. This is also done if you READ A$ from a DATA statement – the pointer points to the BASIC program. However, if you have calculated a string with MID$, CHR$, or A$=A$+B$ etc, then the operating system stores the string in another area of free memory, and the pointer is directed here. If you multiply the contents of byte 5 by 256 and add byte 4, you will find the starting point for the string. Simply count along (Byte 3) number of characters.

After thinking about this for a minute and a half, you should be able to see that you can swap two variables simply by changing the first two bytes. This assumes both variables are the same type. Otherwise you'll just get garbage.

I decided that I wanted to do it differently. Instead of changing the names over, I would change the bytes which contain the variable, bytes 3,4,5,6 and 7. My reasoning was this: if you use a variable heaps of times, it takes less time to find, if it is at the start of the variable table. If you change the name, the variable will jump about all over the place. Changing the data makes the variable stay put.

I ended up with a routine 33 bytes long. But of course, it was crude! You could swap a string with an integer variable, or a floating point with an array and create utter havoc! So, out came the pencil and rubber, and I created a test to ensure the same variable type, with a '?TYPE MISMATCH ERROR' if you were naughty.

Fair enough. Would it work with arrays too? No such luck. It was an utter disaster.

Out came the reference books, and I discovered that array variables are stored differently from simple variables. The 3 zeros in integer variables and the 2 zeros in string variables are left out. When I swapped the five bytes when using an array, I was corrupting more than one variable!

So, the blue pencil struck again, and I added a test for the variable type. If an integer, swap only 2 bytes. If a string, swap 3, and for floating point

variables, swap all 5.

Now we have a routine which will swap two variables of the same type, and even from a string to a string array, or floating point to floating point array.

The implications are staggering – I'm staggered if I can find a use for it!

Seriously, if you have a sorting routine which orders an array and does something like:- T$ B$(n) : B$(n) = M$(m) = T$ then this routine is for you. It is faster, and saves using the extra variable. It probably also helps postpone the dreaded garbage collection, which occurs when you do extensive string manipulations.

Now for a closer inspection.

We want to write SYS location, first variable, second variable.

First we have to check for a comma. We'll use a routine from the operating system for this. Next, find the first variable. The operating system conveniently does this, and leaves a pointer for us to follow indicating where the variable can be found. We'll store this pointer. It also leaves a flag as to which type of variable it is. Store this too.

Check for another comma, and find the next variable. Store it's location pointer.

Compare the variable type flags with those stored from the previous variable. If they are different, send a '?TYPE MISMATCH'.

Now we look to find out what type of variable it is, and get the appropriate number of bytes to swap.

Now comes the easy part.

Get a byte. Save it. Transfer from the second variable to the first. Restore the saved byte. Store it in the second variable.

Do it again if necessary.

DONE!

Now put our theories to the test.

```
BARTER.PRG
BASIC 2.0

0000 20 F8 CD       JSR CDF8     'check for a comma
0003 20 6D CF       JSR CF6D     'search for variable location
0006 85 0F          STA 0F       'store pointer to variable in
0008 84 10          STY 10       'zero page
000A A5 07          LDA 07       'save flags
000C 85 01          STA 01       'which indicate
000E A5 08          LDA 08       'variable type
0010 85 02          STA 02
0012 20 F8 CD       JSR CDF8     'check comma
0015 20 6D CF       JSR CF6D     'search for variable location
0018 85 BE          STA BE       'store pointer in
001A 84 BF          STY BF       'zero page
001C A5 08          LDA 08       'compare type with that of
001E C5 02          CMP 02       'first variable
0020 D0 16          BNE 0048     'different? then error
0022 A5 07          LDA 07       'check other flag
0024 C5 01          CMP 01       'different?
0026 D0 10          BNE 0048     'then error
0028 C9 FF          CMP #$FF     'is it a string?
002A D0 04          BNE 0030     '
002C A0 02          LDY #$02     'yes, so swap 3
002E D0 0A          BNE 003A     'go to swap routine
0030 A5 08          LDA 08       'must be integer of floating point
0032 D0 04          BNE 0038     'branch if integer
0034 A0 04          LDY #$04     'swap 5 for floating point
0036 D0 02          BNE 003A     'go to swap routine
0038 A0 01          LDY #$01     'swap 2 for integer
003A B1 BE          LDA (BE), Y  'get byte from second variable
003C 48             PHA          'store it temporarily
003D B1 0F          LDA (0F), Y
003F 91 BE          STA (BE), Y  'store it in second variable
0041 68             PLA          'retrieve original value
0042 91 0F          STA (0F), Y  'store it in first variable
0044 88             DEY          'decrease counter
0045 10 F3          BPL 003A     'go back if counter not minus yet
0047 60             RTS          'all finished - return
0048 A2 A3          LDX #$A3     'error routine
004a 4C 57 C3       JMP C357     'print type mismatch and ready
```

```
BARTER.PRG
VIC BASIC V2.

0000 20 FD CE       JSR CEFD     'check for a comma
0003 20 8B D0       JSR D08B     'search for variable location
0006 85 A5          STA A5       'store pointer to variable in
0008 84 A6          STY A6       'zero page
000A A5 0D          LDA 0D       'save flags
000C 85 01          STA 01       'which indicate
000E A5 0E          LDA 0E       'variable type
0010 85 02          STA 02       '
0012 20 FD CE       JSR CEFD     'check comma
0015 20 8B D0       JSR D08B     'search for variable location
0018 85 A7          STA A7       'store pointer in
001A 84 A8          STY A8       'zero page
001C A5 0E          LDA 0E       'compare type with that of
001E C5 02          CMP 02       'first variable
0020 D0 16          BNE 0048     'different? then error
0022 A5 0D          LDA 0D       'check other flag
0024 C5 01          CMP 01       'different?
0026 D0 10          BNE 0048     'then error
0028 C9 FF          CMP #$FF     'is it a string?
002A D0 04          BNE 0030     '
002C A0 02          LDY #$02     'yes, so we'll swap 3 bytes
002E D0 0A          BNE 003A     'go to swap routine
0030 A5 0D          LDA 0D       'must be integer or floating point
0032 D0 04          BNE 0038     'branch if integer
0034 A0 04          LDY #$04     'swap 5 for floating point
0036 D0 02          BNE 003A     'goto swap routine
0038 A0 02          LDY #$01     'swap 2 for integer
003A B1 A7          LDA (A7), Y  'get byte from second variable
003C 48             PHA          'store it temporarily
003D B1 A5          LDA (A5), Y  'get byte from first variable
003F 91 A7          STA (A7), Y  'store it in second variable
0041 68             PLA          'retrieve orignal value
0042 91 A5          STA (A5), Y  'store it in first variable
0044 88             DEY          'decrease counter
0045 10 F3          BPL 003A     'go back if counter not minus yet
0047 60             RTS          'all finished - return
0048 A2 16          LDX #$16     'error routine
004A 4C 3A C4       JMP C43A     'print type mismatch and ready
```

```
BARTER.PRG
VIC BASIC V2. HEXDUMP

.:  20 FD CE 20 8B
.:  D0 85 A5 84 A6
.:  A5 0D 85 01 A5
.:  0E 85 02 20 F3
.:  CE 20 8B D0 85
.:  A7 84 A8 A5 0E
.:  C5 02 D0 26 A5
.:  0D C5 01 D0 20
.:  C9 FF D0 04 A0
.:  02 D0 0A A5 0D
.:  D0 04 A0 04 D0
.:  02 A0 01 B1 A7
.:  48 B1 A5 91 A7
.:  68 91 A5 88 10
.:  F3 60 A2 16 4C
.:  3A C4 00 00 00
```

```
BARTER.PRG
BASIC 2.0 HEXDUMP

.:  20 F8 CD 20 6D CF 85 0F
.:  84 10 A5 07 85 01 A5 08
.:  85 02 20 F8 CD 20 6D CF
.:  85 BE 84 BF A5 08 C5 02
.:  D0 26 A5 07 C5 01 D0 20
.:  C9 FF D0 04 A0 02 D0 0A
.:  A5 08 D0 04 A0 04 D0 02
.:  A0 01 B1 BE 48 B1 0F 91
.:  BE 68 91 0F 88 10 F3 60
.:  A2 A3 4C 57 C3 00 00 00
```

```
BASIC OBJECT CODE LOADER -

BARTER.PRG BASIC 2.0
1 REM SCRIBE.PRG V01-02
1000 INPUT"START ADDRESS"; SA
1001 REM THIS IS THE ADDRESS YOU WISH TO LOAD THE ROUTINE INTO
1002 REM
1010 FOR I = 0 TO 9 : CO=0
1020 FOR J = 0 TO 7
1030 READ VA : POKE SA+(8*I)+J, VA
1040 CO=CO+VA : NEXT J
1050 READ SU : IF SU <> CO THEN PRINT"?DATA ERROR IN"; 10000+I*10:END:RUN
1060 NEXT I
1070 REM
1080 REM
10000 DATA  32, 248, 205,  32 109, 207, 133,  15,      981
10010 DATA 132,  16, 165,   7, 133,   1, 165,   8,     627
10020 DATA 133,   2,  32, 248, 205,  32, 109, 207,     968
10030 DATA 133, 190, 132, 191, 165,   8, 197,   2,     1018
10040 DATA 208,  38, 165,   7, 197,   1, 208,  32,     856
10050 DATA 201, 255, 208,   4, 160,   2, 208,  10,     1048
10060 DATA 165,   8, 208,   4, 160,   4, 208,   2,     759
10070 DATA 160,   1, 177, 190,  72, 177,  15, 145,     937
10080 DATA 190, 104, 145,  15, 136,  16, 243,  96,     945
10090 DATA 162, 163,  76,  87, 195,   0,   0,   0,     683

READY.
```

```
BARTER.PRG   DATA BLOCK FOR VIC BASIC V2.

10000 DATA   32, 253, 206,  32, 139, 208, 133, 165,     1168
10010 DATA  132, 166, 165,  13, 133,   1, 165,  14,     789
10020 DATA  133,   2,  32, 253, 206,  32, 139, 208,     1005
10030 DATA  133, 167, 132, 168, 165,  14, 197,   2,     978
10040 DATA  208,  38, 165,  13, 197,   1, 208,  32,     862
10050 DATA  201, 255, 208,   4, 160,   2, 208,  10,     1048
10060 DATA  165,  13, 208,   4, 160,   4, 208,   2,     764
10070 DATA  160,   1, 177, 167,  72, 177, 165, 145,     1064
10080 DATA  167, 104, 145, 165, 136,  16, 243,  96,     1072
10090 DATA  162,  22,  76,  58, 196,   0,   0,   0,     514
READY.

SUBSTITUTE THESE LINES FOR THE DATA STATEMENTS IN
THE BASIC 2.0 VERSION OF THE OBJECT CODE LOADER.
```

HOW TO INSTALL THE ROUTINE

If you are using the BASIC Code Loader, simply input the start address when asked. Suggestions are the cassette buffer:-

PET – enter 826 VIC – enter 828

When you want to call the routine, the procedure would be :-

SYS826,B$,F$
or SYS828,A%,B% etc.

The disadvantage of this method is that you cannot then use the cassette without destroying your routine. Besides, far too many programs already reside in PET's second cassette buffer.

Another method would be to store it at the top of memory. As you know BASIC only has so much memory available – **** BYTES FREE tells you exactly how much. In order to prevent BASIC from using our space, we are going to decrease the amount of memory it can use. This is only temporary - it will still be there when you next power up.

On the VIC, type:-

PRINT PEEK(55)+PEEK(56)*256.

The result is the current 'top of memory'. We want to use 80 bytes, so subtract 80 from your top of memory. I'll call this value MEMTOP.

PET users do the same:- PRINT PEEK(52)+PEEK(53)*256. Subtract 80. Now, type:- A=INT((MEMTOP)/256) :B=(MEMTOP)-256*A.

VIC users:- POKE55,B:POKE56,A

PET users:- POKE52,B:POKE53,A

This should lower your top of memory by 80 bytes.

(We actually only need 77 bytes, but using 80 makes the BASIC code loader easier to write.)

If you own a PET that has a built-in Machine Code Monitor, you could enter the values from the Hexdump. VIC owners with the Vicmon Monitor Cartridge could do likewise. Those of you with assemblers should be able to follow the disassembly. Remember that if you enter the code by these methods, you must still protect it, because if BASIC can get it's hands on your work, it will devour it body and soul. (BASIC is very vindictive that way - I hope you keep yours securely chained in it's kennel er ...or is that kernal?)

RELATIVE FILES

by Paul Blair
PET Users Group, ACT

By now, users with BASIC 4.0 CPU's and DOS 2 disk drives will be experimenting with relative files. I don't propose to go over the mechanisms of using relative files.....this has been adequately documented in a number of freely available books and journals. The main point to remember, however, is that relative file handling is built around a unique relative record reference number, the number being used for read/write operations.

My examination of relative files came about because of a request for a simple listing program for about 1000 items. The program is to run in real time, with frequent input, deletion and amendment of records. The listing had to be available for sorted output at any time to the printer. I could cope with the first requirements, but the sorted output requirement seemed a considerable difficulty.

My first thoughts ran to some form of disk sort, but with 1000 items and a fast sorting requirement, it was not a practical solution. As the sort was a prime requirement, I thought about that aspect next, and decided to create a descriptor file that could be sorted easily, and which could be used to control operations of the relative files. The descriptor (or index) file would contain a marker and a unique record number. The marker would indicate whether the particular relative record is 'occupied' by an active record, or 'vacant' available for a new record. The marker chosen to indicate 'vacant' record is the '[', because this character follows 'Z', and will sort 'vacant' descriptors to the end of the file.

To illustrate what all this means, load and run FILE MAKER VI. This program sets up the appropriate sequential file, and writes 50 blank relative records to disk. The sequential file looks like this:-

```
[001
[002
[003
and so on to Record #50
[050
```

As explained above, the left square bracket '[' is the marker to indicate that the record number following it is 'vacant'.

Now load 'USERV4', and RUN it. From the menu, we decide to enter a new record, Fred Nurk. The key field is given as NURK, and this will be used in later operations.

Addition of a new item causes a search for the first occurence of a '[' character in the sequential descriptor file .....say it finds '[016'. The program takes the key field, and this key field replaces the '['. Using NURK as the key field, Fred's descriptor file entry would read 'NURK016'. The program then writes Fred's details to Record #16, and the descriptor file is sorted .....about 3 seconds for a 1000-entry file.

Having a sorted list, the program can find Fred again very quickly using a binary chop search on all or part of his key field. Typical search time would be less than half a second, and disk access takes another second or two. So, any record can be listed to the screen in not more than a few seconds. A binary chop search? .....ASK, if this is new to you.

Deletion is also very easy .....the descriptor file entry is returned to its original '[016' (so we will know that Record #16 is once more available) but the relative record is not erased. It is left on disk until the next entry, which overwrites the old record.

The final programs are not complete, but the development programs FILE MAKER VI and USERV4 are, warts and all! FILE MAKER has been explained. USERV4 is the (still clumsy) file handler, not very pretty or tidy but it seems to work. Both programs require BASIC 4.0 and DOS 2.

For demonstration purposes, each relative record is 60 characters long, made up of three sub-fields. The usual partial record handling techniques could be built into USERV4, if so desired, greatly enhancing data handling.

GOTCHAS: There are no handy 'escape routes' built in as yet, so be careful if you 'break out' of the program.

FILE MAKER V1

```
100 REM"      ┌──────────────┐
110 REM"      | MASTERFILE   |
120 REM"      | 16 MAY 82    |
130 REM"      └──────────────┘
140 R$=CHR$(13):OPEN2,8,2,"0:MASTERLIST,S,W":
    GOSUB260
150 FORT=1TO50:T$=STR$(T):T$=RIGHT$
    (T$,LEN(T$)-1):T$=RIGHT$("000"+T$,3)
160 T$="["+T$:PRINT#2,T$","R$;
170 NEXT:CLOSE2:GOSUB260
180 REM"      ┌──────────────┐
190 REM"      | RELATIVE FILE |
200 REM"      └──────────────┘
210 DOPEN#1,"MASTER FILE",D0,L60:GOSUB260
220 RECORD#1,50:GOSUB260
230 PRINT#1,CHR$(255);:GOSUB260
240 DCLOSE#1
250 END
260 IFDS<20THENRETURN
270 PRINTDS$
280 IFDS=50THENRETURN
290 STOP
READY.
```

```
100 REM"     ┌──────────────┐
110 REM"     ¦ MASTERUSER   ¦
120 REM"     ¦ 16 MAY 82    ¦
130 REM"     └──────────────┘
131 ONSNGOTO132,134
132 SN=2:LOAD"SUPERSORT4.$7A97",8
134 CLR:POKE53,122:POKE49,122:POKE52,150:POKE48,150
135 LN$="-":FORA=1TO6:LN$=LN$+LN$:NEXT:LN$=LEFT$(LN$,39)
140 GOSUB14000:DIMMF$(60):R$=CHR$(13):BL$=" ":FORA=1TO6:BL$=BL$+BL$:NEXT
150 OPEN2,8,2,"0:MASTERLIST,S,R":GOSUB2000:A=-1
160 A=A+1:INPUT#2,MF$(A):TS=ST:IFLEFT$(MF$(A),1)="["THENTL=TL+1
165 IFTS=0THEN160
170 CLOSE2:PRINT"■"A+1"RECORDS IN FILE"
180 PRINT"■"TL"RECORDS UNUSED":PRINT"■"LN$:GOSUB1500:GOSUB10000
187 REM"     ┌──────────────┐
188 REM"     ¦ MAIN MENU    ¦
189 REM"     └──────────────┘
190 HD$="MAIN MENU":GOSUB17000:PRINT"1...TO ADD TO FILE":PRINT"■2...TO DELETE"
200 PRINT"■3...TO FIND RECORD":PRINT"■4...TO AMEND":PRINT"■5...TO PRINT"
210 PRINT"■6...TO QUIT":PRINTLN$:PRINT"■PLEASE MAKE YOUR CHOICE"
220 GETAN$:IFAN$=""THEN220
230 AN=VAL(AN$):IFAN<1ORAN>6THEN220
240 ONANGOSUB4000,5000,6000,7000,8000,20000:GOTO190
1497 REM"    ┌──────────────┐
1498 REM"    ¦ WAIT A BIT   ¦
1499 REM"    └──────────────┘
1500 POKE158,0:PRINT"■PRESS ANY KEY TO CONTINUE"
1510 GETAN$:IFAN$=""THEN1510
1520 RETURN
1997 REM"    ┌──────────────┐
1998 REM"    ¦ DISK ERROR   ¦
1999 REM"    └──────────────┘
2000 IFDS<20THENRETURN
2010 PRINTDS$
2020 IFDS=50THENRETURN
2030 STOP
2100 FORDL=1TO2000:NEXT:RETURN
3997 REM"    ┌──────────────┐
3998 REM"    ¦ ADD TO FILE  ¦
3999 REM"    └──────────────┘
4000 FL=1:HD$="ADD A RECORD":GOSUB17000:GOSUB10000:FORUY=0TO49
4005 IFLEFT$(MF$(UY),1)="["THENID$=RIGHT$(MF$(UY),3):ID=VAL(ID$):GOTO4020
4010 NEXT
4020 PRINT"ENTER KEYNAME ":P3=15:GOSUB12000:PRINT"■YOUR KEY IS: "P$:J$=P$
4023 MF$(UY)=P$+ID$
4025 PRINT"■ENTER DETAILS ":P3=35:GOSUB12000:PRINT"■YOUR ENTRY IS: "J$+" "+P$
4040 P$=J$+" "+P$:UY=ID:GOSUB13000:GOSUB1500:RETURN
4997 REM"    ┌──────────────┐
4998 REM"    ¦ DELETE FILE  ¦
4999 REM"    └──────────────┘
5000 HD$="DELETE":GOSUB17000:GOSUB6010
5010 PRINT"■TYPE ■D■ TO DELETE ■M■ FOR MENU"
5020 GETAN$:IFAN$=""THEN5020
5030 IFAN$="M"THEN190
5040 IFAN$="D"THENFL=1:GOTO5060
5050 GOTO5020
5060 MF$(ZZ)="["+ID$:GOSUB1500:RETURN
5997 REM"    ┌──────────────┐
5998 REM"    ¦ FIND RECORD  ¦
5999 REM"    └──────────────┘
6000 HD$="SEARCH":GOSUB17000
6010 PRINT"WHICH RECORD PLEASE? ";:P3=12:GOSUB12000:ZT$=P$:GOSUB9000
6020 GOSUB11000:GOSUB2100:RETURN
6997 REM"    ┌──────────────┐
6998 REM"    ¦ AMND RECORD  ¦
6999 REM"    └──────────────┘
7000 HD$="AMEND RECORD":GOSUB17000:PRINT"WHOSE RECORD TO AMEND? ";:P3=15
7010 GOSUB12000:ZT$=P$:GOSUB9000:UY=ZZ:PRINT"■AMEND TO- "
7020 GOSUB4020:PRINT"■FIXED":GOSUB2100:FL=1:RETURN
7997 REM"    ┌──────────────┐
7998 REM"    ¦ PRINT FILES  ¦
7999 REM"    └──────────────┘
8000 HD$="LIST OF RECORDS":GOSUB17000:GOSUB10000
8010 PRINT"PRESS ANY KEY TO SEE LIST OF RECORDS":PRINTLN$"■"
8020 GETAN$:IFAN$=""THEN8020
8030 AR=0:FORUY=0TO50:AB$=LEFT$(MF$(UY),1):IFAB$="["ORAB$=""THEN8050
```

```
8040 PRINTRIGHT$(MF$(UY),3)"......."LEFT$(MF$(UY),LEN(MF$(UY))-3):AR=AR+1
8050 NEXT:PRINT"■"LN$:PRINT"■TOTAL USED RECORDS "AR:PRINT"■"LN$
8060 GOSUB1500:RETURN
8997 REM"       ┌──────────┐
8998 REM"       | CHOP SEARCH |
8999 REM"       └──────────┘
9000 GOSUB10000:ZT=0:ZB=49
9010 ZM=ZT+INT((ZB-ZT)/2):ZI$=MF$(ZM):GOSUB9160
9020 IFZI$=ZT$THEN9060
9030 IFZT=ZMTHEN9150
9040 IFZI$=""ORZT$>ZI$THENZT=ZM:GOTO9010
9050 ZB=ZM:GOTO9010
9060 ZB=ZM
9070 IFZB-1<0THEN9100
9080 ZI$=MF$(ZB-1):GOSUB9160:IFZI$<>ZT$THEN9100
9090 ZB=ZB-1:GOTO9070
9100 ZT=ZM
9110 IFZT+1>ATHEN9140
9120 ZI$=MF$(ZT+1):GOSUB9160:IFZI$<>ZT$THEN9140
9130 ZT=ZT+1:GOTO9110
9140 FORZZ=ZBTOZT:PRINT"■"LEFT$(MF$(ZZ),LEN(MF$(ZZ))-3)
9144 NEXT
9146 IFZZ=0THENID$=RIGHT$(MF$(ZZ),3):ID=VAL(ID$):GOTO9148
9147 ZZ=ZZ-1:ID$=RIGHT$(MF$(ZZ),3):ID=VAL(ID$)
9148 RETURN
9150 PRINT"■ NO MATCH FOUND ":RETURN
9160 IFLEN(ZI$)>LEN(ZT$)THENZI$=LEFT$(ZI$,LEN(ZT$))
9170 RETURN
9997 REM"       ┌──────────┐
9998 REM"       | SORT IT ALL |
9999 REM"       └──────────┘
10000 POKE905,1:POKE907,1:POKE906,1:POKE927,5:POKE947,0:SYS31383:RETURN
10997 REM"      ┌──────────┐
10998 REM"      | ACCESS FILE |
10999 REM"      └──────────┘
11000 DOPEN#1,"MASTER FILE",D0:GOSUB2000:RECORD#1,(ID):INPUT#1,IN$
11010 PRINT"■RECORD IS-":PRINT"■■"IN$:GOSUB2100:DCLOSE#1:RETURN
11997 REM"      ┌──────────┐
11998 REM"      | INPUT INPUT |
11999 REM"      └──────────┘
12000 P$="":FOR0=1TOP3:PRINT"▒";:NEXT:FOR0=1TOP3:PRINT"▌";:NEXT
12010 GETA$:IFA$=""THEN12010
12020 IFA$=CHR$(13)THENPRINTLEFT$(BL$,P3-LEN(P$)):RETURN
12030 IFA$="←"THEN12070
12040 IFA$<>CHR$(20)THEN12100
12050 IFP$=""THEN12000
12060 FOR0=1TOLEN(P$):PRINT"▌";:NEXT:GOTO12000
12070 IFP$=""THEN12010
12080 IFLEN(P$)=1THENP$="":GOTO12060
12090 P$=LEFT$(P$,LEN(P$)-1):PRINT"▌▒▌";:GOTO12010
12100 IFA$<" "OR(A$)"Z"ANDA$<"▌")THEN12010
12110 IFP3=1THENP$=A$:PRINTP$CHR$(13);:RETURN
12120 IFLEN(P$)=P3THENPRINTCHR$(13);:RETURN
12130 P$=P$+A$:PRINTA$;:GOTO12010
12997 REM"      ┌──────────┐
12998 REM"      | OUTPUT FILE |
12999 REM"      └──────────┘
13000 DOPEN#1,"MASTER FILE",D0:GOSUB2000
13010 RECORD#1,(UY):P$=LEFT$(P$+BL$,50):PRINT#1,P$:GOSUB2000
13020 DCLOSE#1:GOSUB2000:RETURN
13997 REM"      ┌──────────┐
13998 REM"      | PRETTY PAGE |
13999 REM"      └──────────┘
14000 PRINT"■_____":PRINT"■ RELATIVE FILES ■":PRINT"■V2-18 MAY 82■
14010 PRINTLN$:RETURN:REM ROOM FOR MORE
16997 REM"      ┌──────────┐
16998 REM"      | PAGE HEADER |
16999 REM"      └──────────┘
17000 PRINT"■";:MM=(LEN(HD$))/2:PRINTTAB(20-MM)HD$:PRINTLN$:RETURN
19997 REM"      ┌──────────┐
19998 REM"      | FINISH OFF |
19999 REM"      └──────────┘
20000 IFFL=0THENPRINT"■":END
20010 OPEN2,8,2,"@0:MASTERLIST,S,W":GOSUB2000
20020 FORV=0TOA:PRINT#2,MF$(V)","R$;
20030 NEXT:CLOSE2:GOSUB2000:PRINT"■":END
READY.
```

COMMODORE 9060/9090 HARD DISK DRIVE

by Ian Webster

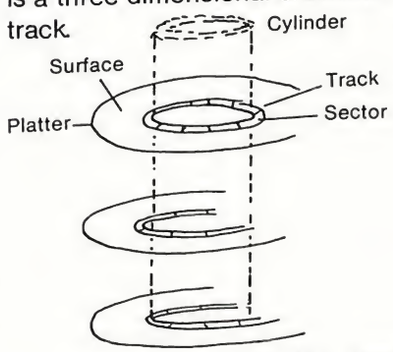
Commodore has announced the release of two hard disk drive peripherals for the CBM range of computers. These drives will provide a significant increase in the power and versitility of CBM computers wherever large data storage, speed of data access and the integrity of large applications software are required. Lets take a look at these new peripherals.

HARD DISK DRIVE TECHNOLOGY

Hard disk drive technology is completely different to that used in floppy disk drives. A hard disk consists of a cartridge containing one or more solid platters mounted on a spindle. Each platter is similar to an LP record. This cartridge, either permanently fixed or removeable, is mounted within a disk drive. The platters rotate at very high speed and retractable heads, one for every surface in the cartridge, are floated across the surface of the disk. The heads are aerodynamically designed to float on a very thin cushion of air generated by the rapidly spinning disk. The term 'disk drive head crash' actually refers to the floating head either bouncing off the surface of the disk or in extreme cases crashing into the surface of the disk and scratching the surface.

Winchester technology solved the problems involved in building small hard disk systems that could be economically manufactured to withstand the rigours of microcomputer environments. These disk units typically use a fixed cartridge of two to four platters either 8" or 5" in diameter. The cartridge and head assembly are enclosed in a sealed unit to eliminate the possiblity of foreign particles contaminating the disk surface.

Typical data capacity of 5" drives is 5 to 10 Megabytes and 10 to 25 Megabytes for 8" drives. Data capacity is usually given as unformatted capacity which is the capacity of the drive to encode data of any type, while formatted capacity is the capacity of the drive to accept blocks of data after the operating system has written the block formatting information onto the disk. The blocks on a hard disk are organised as sectors, tracks and cylinders. A sector is a block of data, usually 256 or 512 bytes of information. Sectors are organised into tracks, which refer to all the sectors at a particular radius of the disk on one surface of the disk. A cylinder consists of all particular tracks on all surfaces of the disk. This means that cylinder 5 contains all of the sectors in track 5, surface 1, surface 2, surface 3 and so on. The term track is not often used with hard disk terminology, as it is the cylinder that is referenced by the disk operating system. In fact the cylinder is a three dimensional extension of a track.



Sectors are usually mapped continuously within each cylinder, so that sector 1, cylinder 5 would be located on the first surface, sector 33 on the second surface, sector 65 on the third surface and so on until all the surfaces have been accessed.

COMMODORE HARD DISK DRIVES

The Commodore hard disk drives are built around units sourced from Tandon. The Tandon Winchester unit is built into a Commodore chassis approximately 6" x 8" x 15" together with Commodore's IEEE interface electronics. Unfortunately for the voyers, the Tandon Winchester drive is sealed in a steel box, so the action of the drive is not visible.

The platters spin continually at 3600rpm, more than 10 times faster than the 300rpm of a CBM 8050 disk drive. Track to track access time is 3 milliseconds, much faster than the 30 milliseconds of an 8050. The start up delay of floppy disk drives is also eliminated as the hard disk drive is always rotating.

THE CBM 9060 HARD DISK DRIVE

The CBM 9060 drive is based on a Tandon 601 hard disk unit. This unit has three platters with a head for each surface of the disk cartridge. The unformated capacity of the unit is 6.38 Megabytes. After formatting with 32 sectors per track and 153 cylinders of 6 tracks data capacity is 19442 blocks or 4.97 Megabytes.

THE CBM 9090 HARD DISK DRIVE

The CBM 9090 drive is based on a Tandon 602 hard disk unit. This unit has four platters with two heads per platter. The unformatted capacity is 9.57 Megabytes and the capacity after 153 cylinders of 8 tracks with 32 sectors per track have been formatted is 7.46 Megabytes.

The performance specification of both drives is identical.

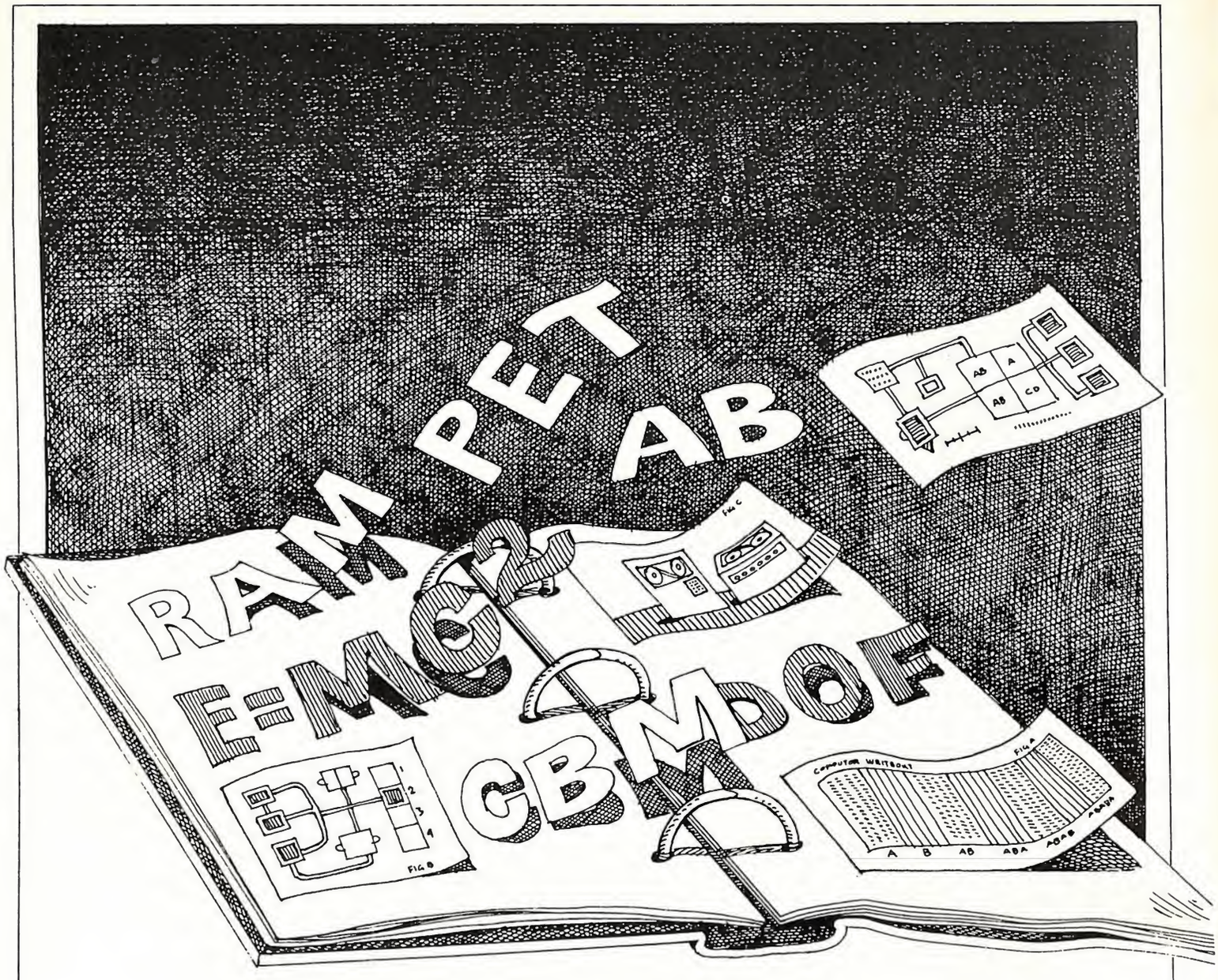
HARD DISK DRIVE INTERFACING

The disk drive is interfaced to the IEEE bus similar to any other Commodore peripheral device. The device number of the drive defaults to unit 8, drive zero but can be changed to any other device address using the standard technique. The drive cannot be configured into multiple logical drives and must be used as one drive.

HARD DISK DRIVE SOFTWARE

The hard disk drives use Commodore DOS 3.0 often referred to as

Continued on Page 30



Excerpts from a Technical Notebook

Bits & Pieces

***Faster Than A Speeding Cathode Ray!**

```
10 PRINT"[CLR DN DN]IS NOT YOUR NAME
   ABE LINCOLN?":GOTO10
```

The string in the above statement prints and clears so fast that the screen can't keep up! You might expect the text to 'flash' on and off. But, as the trace is scanning the screen, the text actually prints, clears, and prints again before the trace gets a chance to erase. It's hard to say how many, but BASIC prints and clears several times during a single screen scan. Therefore, the text appears to be stationary, as if the Clear Screen character was not even there!

Then, they become 'un-synchronized.' At this point, the text appears to be drawn slowly across the line. The trace draws part of the text and then it's turned off again by the Clear Screen. The same thing happens next time around only a little farther to the left or right. It's rather hard to explain but not hard to imagine when you're looking at it.

Try different combinations by adding or removing CLRs, DNs, characters, commas and semicolons. For an interesting effect, add line 20 by simply duplicating line 10 (remove the GOTO 10 and add it at the end of line 20). Try this one too:

```
10 PRINT"[CLR 6DN]IS YOUR NAME ABE
   LINCOLN?";:GOTO10
```

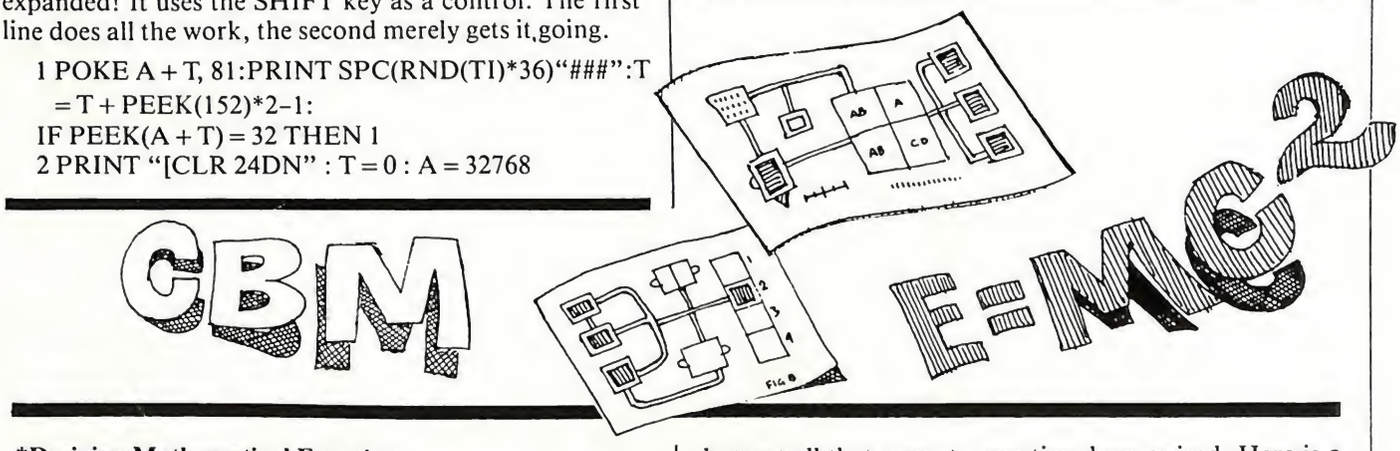
6DN = 6 cursor downs. Different machines produce different results. These were done on forty column PETs. 80 column users will have to modify the statements slightly to get the right effect since the scan speed is somewhat different.

Richard also has a one-line game which surely could be expanded! It uses the SHIFT key as a control. The first line does all the work, the second merely gets it going.

```
1 POKE A+T, 81:PRINT SPC(RND(TI)*36)"###":T
  =T+PEEK(152)*2-1:
IF PEEK(A+T)=32 THEN 1
2 PRINT "[CLR 24DN" : T=0 : A=32768
```

***More One-Liners**

```
1 FOR X=0 TO 999 : POKE 32768+X,
      (PEEK(32768+X)+128)AND255:NEXT
1 a=32768:i=0:j=38
2 S=SGN(J-I) : FOR X=I TO J STEP S : POKE A
      +X, 32 : POKE A+X+S, 87 :
NEXTX : I=39-I : J=39-J : GOTO 2
```



***Deriving Mathematical Functions**

BASIC has some trigonometric functions implemented but not all that may at some time be required. Here is a handy list:

Secant	SEC(X) = 1/COS(X)
Cosecant	CSC(X) = 1/SIN(X)
Cotangent	COT(X) = 1/TAN(X)
Inverse Sine	ARCSIN(X) = ATN(X/SQR(-X*X+1))
Inverse Cosine	ARCCOS(X) = -ATN(X/SQR(-X*X+1))+π/2
Inverse Secant	ARCSEC(X) = ATN(X/SQR(X*X-1))
Inverse Cosecant	ARCCSC(X) = ATN(X/SQR(X*X-1))+(SGN(X)-1*π/2
Inverse Cotangent	ARCCOT(X) = ATN(X)+π/2
Hyperbolic Sine	SINH(X) = (EXP(X)-EXP(-X))/2
Hyperbolic Cosine	COSH(X) = (EXP(X)+EXP(-X))/2
Hyperbolic Tangent	TANH(X) = EXP(-X)/(EXP(X)+EXP(-X))*2-1
Hyperbolic Secant	SECH(X) = 2/(EXP(X)-EXP(-X)
Hyperbolic Cosecant	CSCH(X) = 2/(EXP(X)-EXP(-X)
Hyperbolic Cotangent	COTH(X) = EXP(-X)/(EXP(X)-EXP(-X))*2+1
Inverse Hyperbolic Sine	ARCSINH(X) = LOG(X+SQR(X*X+1))
Inverse Hyperbolic Cosine	ARCCOSH(X) = LOG(X+SQR(X*X-1))
Inverse Hyperbolic Tangent	ARCTANH(X) = LOG((1+X)/(1-X))/2
Inverse Hyperbolic Secant	ARCSECH(X) = LOG((SQR(-X*X+1)+1/X))
Inverse Hyperbolic Cosecant	ARCCSCH(X) = LOG((SGN(X)*SQR(X*X+1/X))
Inverse Hyperbolic Cotangent	ARCCOTH(X) = LOG((X+1)/(X-1))/2

PRINTER FORMATTING

by Garry Mason

As you may know all CBM printers are intelligent; this means that along with being able to print in normal mode, they can be setup to perform formatting. For example, this could be used to produce forms or reports with beautiful columns all correctly lined up, without 10K of program text to get it to work.

Before we go into this subject any further we should first become acquainted with the idea of secondary addresses and how they work.

On the IEEE bus each listening device must be given a number. This is used when communicating to make sure that devices on the bus get only the information directed at them. All Commodore printers for example have a device number of 4.

So when the IEEE controller (the computer), starts to talk to any device, it firstly puts the device number on the bus and waits for a device to respond. If no device responds in 65 milliseconds DEVICE NOT PRESENT will occur. Once the device has responded data transfer can continue. The data transfer being prefixed in all cases by transferring a special control character, this being the secondary address. In some cases this is ignored, in Commodore devices this is used to signal to the micro-processor that something special is going on.

'Special' means a lot in Commodore's case. One of the most useful is to control the formatting of text. The way this works is to supply the printer with a mask into which it can put data. This means that the data can be sent to the printer without any special organisation. This becomes very useful in putting variables into money format, such as converting 10.9345 to $10.93.

The way in which the mask is sent is quite simple. The first thing to do is to OPEN a file to the printer and set the correct secondary address, thus indicating that a mask will follow under this address.

```
10  OPEN 1,4,2
```

The first digit is the file number, followed by the device number and then the secondary address. This means that all references to logical file 1 are destined for device 4 and secondary address 2. Secondary address 2 being the factor which tells the printer that a format mask is about to follow.

Now if we execute:-

```
20  PRINT # 1, " 99.99 AAAAA"
```

The CBM will send the mask which will dictate how further information is to printed.

To use printer formatting, the secondary addresses we need to know are:-

0 meaning, do not put this data through the present mask

1 meaning, print the following data and use the mask to format the data

2 meaning, here comes the mask.

With our example we now choose to either use the mask or not, by selecting which secondary address we send data to the printer on. This line will prepare the logical files for use, and prints the header and an example:-

```
30  OPEN 2, 4, 0 : OPEN 3, 4, 1
40  PRINT #2, "THIS COULD BE A
    REPORT TITLE"
50  PRINT #3, 54.5894,
    "IAMACOMPLETEFRUITCAKE"
60  CLOSE 1 : CLOSE 2 : CLOSE 3
```

Defining the mask to suit your needs can be a joy once one has a grasp of the symbols used. All the symbols do is to tell the processor how to match the incoming data with the mask. Different symbols mean different ways of inserting the data into a mask. Anything else but a control symbol is called a litteral, the litterals in the above example were the space characters. This is because these spaces will always be printed as spaces no matter how the data is aligned. Other literals need to be preceeded by the RVS symbol.

SYMBOL CHART

A The letter 'A' represents any character or digit. A field of this type would be left-justified and would be padded on the right of the field with blanks. Blanks can be made to appear by using SHIFTED spaces, instead of spaces. Literals cannot be printed in the middle of a field.

9 This is the general numeric mask used for all numerics (no letters). Any letter in this field will cause the format of the whole line to break up. If a '9' is present after the decimal point in a field, then a zero will be printed if no digit is sent. All unused digits before the decimal point will print a blank.

Z This character is also for displaying numerics. This, unlike the '9' symbol, will force a 0 to be printed if there is no digit available in that position. It can be used to force a leading zero.

$ The use of a dollar symbol is appropriate when money format is required, as a dollar symbol will be forced next to the field. Note only one dollar will be printed even if several are used. The dollar symbol will be right justified in the field.

S When this character is used, the sign of the number, either '-' or '«' will be printed in that fixed column. This cannot be used at the end of a field.

- This symbol will cause a minus sign to be printed in the fixed column, if the number is

negative. A blank will be substituted if the data number is positive.

Decimal points are used to line up numeric fields under a decimal point. A decimal point may occur at any point in the field. If two are used; a new field will be designated upon meeting the second decimal point. Decimal points cannot be put in alphanumeric fields.

The following table should give an example of the results obtained by the use of formatting control.

FORMATTING EXAMPLES.

FORMAT FIELD:	DATA:	EDITED RESULT:
AAAAA	ABC	ABC
AAAAA	ABCDEFG	ABCDE
$$$$$	99	$99
$9999	99	$ 99
$99.99	77	$77.00
$99.99	-77	$77.00
$99.99-	-77	$77.00-
$$9.99	77	$77.00
S$99.99	77	+$77.00
ZZZZ	77	0077
ZZ.999	.77	00.77
ZZZ.99	77	077.00
.99	77	.**
.99	.001	.00
S.999	1.5E-02	+.015
Z.999-	1.5E-02	0.015
Z.999-	-1.5E-02	0.015-

The printer software handles up to ten significant figures, and an exponent range ± 99 on numbers passed as data to be formatted. Exponential numbers must be normalised so that numbers fall into the range 0 <= x < *10*.

If a numeric is to be sent to a field which has too few character digits then the whole field will be blanked out with asterisks.

Other characters which remain the same on each line (literals) can be printed by preceeding the literal with the reverse-on control character. This allows one to construct forms which use the CBM graphics characters to form the lines.

Placing commas and semi-colons in the PRINT # statement that sends the data to the 'mask' can be very important. Generally commas or semi-colons may follow a numeric field, but a skip character may not. For strings, a skip character must follow a string otherwise the format line will be corrupted. Commas, etc will be ignored at the end of a string so a skip character is a must.

If for some reason you wish to skip a certain field completely, then two skip characters must be included. Compare the following two examples:-

```
SK$=CHR$(29)
PRINT#1,"AAAA AAA AA"
PRINT#3, "FRED" SK$ "ALF" SK$
    "GUNBY" SK$
PRINT#3, "FRED" SK$ SK$
    "GUNBY" SK$
```

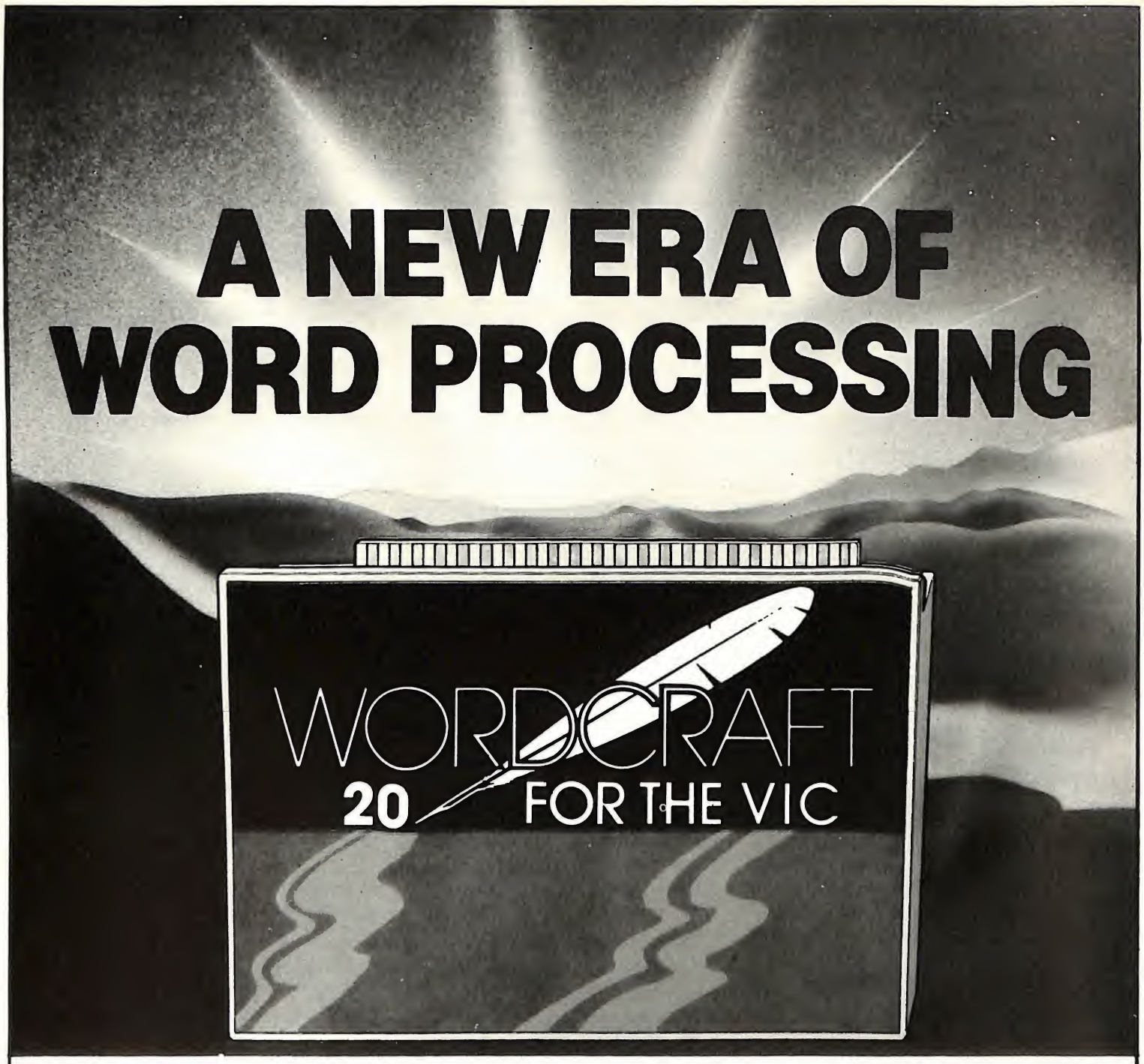
produces on the printer:-

```
FRED        ALF        GU

FRED                   GU
```

The skip character is CHR$(29)

A NEW ERA OF WORD PROCESSING



The introduction of Wordcraft 20 for the VIC brings the benefits and advantages of full scale word processing directly to the general public.
Until now only the business world could afford word processing systems but this amazing price breakthrough makes it available to everyone.
Wordcraft 20 comes on a cartridge ready to plug into the back of the VIC. Included in the cartridge is an extra 8K of RAM that is also available for use with other programs – so not only do you get a word processor but you also get a memory expansion thrown in. The system also comes with complete documentation catering both for the inexperienced user and for those already familiar with Wordcraft 80.
Just look at these features:

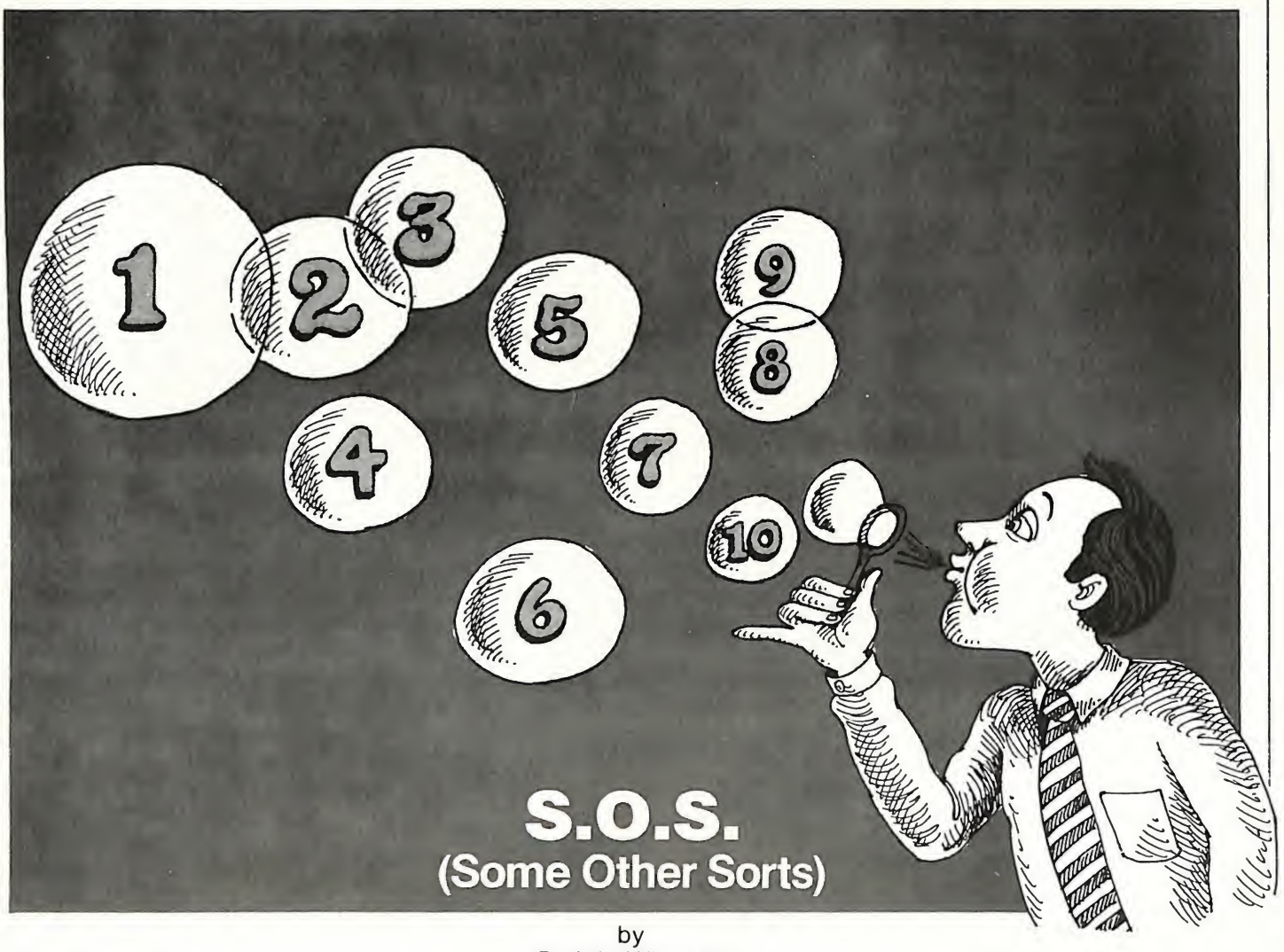
★ Full use of colour and sound.
★ Full compatibility with VIC 1515 printer, parallel printers or RS232C serial printers.
★ Full control over margins, document width, tab stops, decimal tabs, justified output, multiple copies. Complete control of the final output.
★ Automatic underlining and emboldening.
★ Full screen display with automatic paging.
★ Full storage and retrieval facilities from disk and tape.
★ Full compatibility with Wordcraft 80.
★ Name and address capabilities – including labels.
★ Full document merging facilities.

Wordcraft 20. The package that the VIC user has been waiting for. A word processor of proven quality at a low price.
For the first time ever, every home can have one.

CW ELECTRONICS

THE VIC CENTRE

416 LOGAN RD.(Pacific Hwy) STONES CORNER, BRISBANE. TEL: (07) 397 0808, 397 0888 P.O. Box 274, SUNNYBANK QLD.4109. TELEX AA40811



by
Dwight Wheeler

It's been estimated (I don't remember by whom) that of all computer operations, about 25% of the time is spent on sorting. If my own experience is any indication, I would say that figure is probably accurate, or maybe too low. In any case, sorts are indispensable and the faster they run, the better.

You may already be familiar with the bubble sort, the earliest "exchange" sort. While easy to code, such a sort suffers from inefficiency. It will serve for small lists, but when the job gets tougher, we should look elsewhere. So, here's a slightly different sort which adjusts the lower boundary according to the status of the list to be sorted.

I call this the "BBSORT" which is short for "Boundary Bubble Sort." As with all bubble sorts, the heaviest items will sink to the bottom while the light ones rise to the top. What makes this sort different is that it checks the position in the list at which the *last exchange* is made and that point becomes the new lower limit of the next pass. Thus, if the list is in order at the start, no exchanges are made, the lower boundary becomes 0, and the sort stops after only one pass. Since most lists are usually in partial order at the outset, this procedure can mean a considerable saving in time.

Let's look at the code:

```
100 REM  --> PGM: BBSORT <--
```

Since this sort is done in internal memory, we must set up an array. A dimension (DIM) statement is necessary for lists greater than 10 items. The maximum of 100 here is arbitrary:

```
110 DIM A$(100)
120 REM.....LOAD ARRAY.....
```

Set the counter to 1:

```
130 N = 1
```

You could obtain your list for the array directly from the keyboard, or from data files on disc or cassette, or from DATA statements in the program, depending upon your application. For this illustration we'll use DATA statements as the source. DATA can be placed anywhere in the program, but I usually put it last so it can be expanded without conflict in line numbers. Let's read the DATA:

```
140 READ A$(N)
```

Check for the "end" of the data:

```
150 IF A$(N)="END"  THEN 180
```

Add one to the counter:

```
160 N = N + 1
```

Go back and read another data item:

```
170 GOTO 140
```

When all the data have been read, line 150 will detect it and jump down to the actual sort routine. But first we must subtract 1 from the number of items in the array to drop the "END" item which should not be included in the list.

```
180 N = N - 1
190 REM.....SORT.....
```

Here are the "working parts" of this sort. At the beginning, we set the lower boundary (B) at the number of items in the array (N) so that each entry will be looked at at least once.

```
200 B = N
```

Then we immediately check to see if there is anything in the list. If not, the procedure stops at once:

```
210 IF B = Ø THEN 33Ø
```

If B is not equal to zero, the program proceeds. At this point we must establish an intermediate variable as a position locator—call it J—and set it to the top of the list, position 0:

```
220 J = Ø
```

Now we step through the list from 1 to the end minus 1 with a FOR/NEXT loop using the index I:

```
230 FOR I = 1 TO B-1
```

Compare the first item [A$(I)] with the next [A$(I+1)].

```
240 IF A$(I) <= A$(I+1) THEN 290
```

If the first item is less than or equal to the second, it is skipped by jumping to the next I in line 290. If not, the two items are exchanged using a standard three-step exchange routine:

```
249 REM.....EXCHANGE.....
```

A temporary variable T$ is set equal to the first item:

```
250 T$ = A$(I)
```

Then the first item is assigned the value of the second (smaller) item:

```
260 A$(I) = A$(I+1)
```

The second item is then assigned the value of the temporary variable and the exchange is complete:

```
270 A$(I+1) = T$
```

So far so good, but here the plot thickens. Remember "J"? That's the position locator which keeps track of the number in the list where the exchanges are made. In this case, an exchange has been made at position I, so we take note of that:

```
280 J = I
```

Then we go to the next item:

```
290 NEXT I
```

The program goes back to line 230, the index I is incremented by one, and the next two items are compared, etc. Each time an exchange is made, J takes on the value of the index I at that point. When the whole list has been scanned, J will have the value of the index where the last exchange was made. If no exchanges were made, J will equal zero since it was initialized to zero in line 220.

When the FOR/NEXT loop has been completed (at B-1), control drops through to line 300 which sets the lower boundary B to the new value J (which was the point of the last exchange):

```
300 B = J
```

Then we hop back up to see if the new boundary is zero.

```
310 GOTO 210
```

If B is not equal to zero, the list is run through again—but only as far as the new lower boundary.

The loop is repeated until no exchanges are made J = 0 = B). At this point, line 210 will transfer control to the next segment of the program—in this case, the printout.

```
320 REM.....PRINT.....
330 FOR X = 1 TO N
340 PRINT A$(X);"  " ;
350 NEXT X
360 END
```

Finally, here is the long-lost DATA statement. As a handy "benchmark" for comparing sorts, I often use the sequence on the keyboard:

```
370 DATA Q,W,E,R,T,Y,U,I,O,P,A,S
    D,F,G,H,J,K,L,Z,X,C,V,B,N,M
```

Now, the moment of truth . . . run it, and count the number of seconds it takes to sort. Just for fun, change the DATA line to place all the letters in alphabetical order. Run it again . . . surprise? It prints almost immediately! So our "Boundary Bubble Sort" works and is an improvement on the average bubble sort.

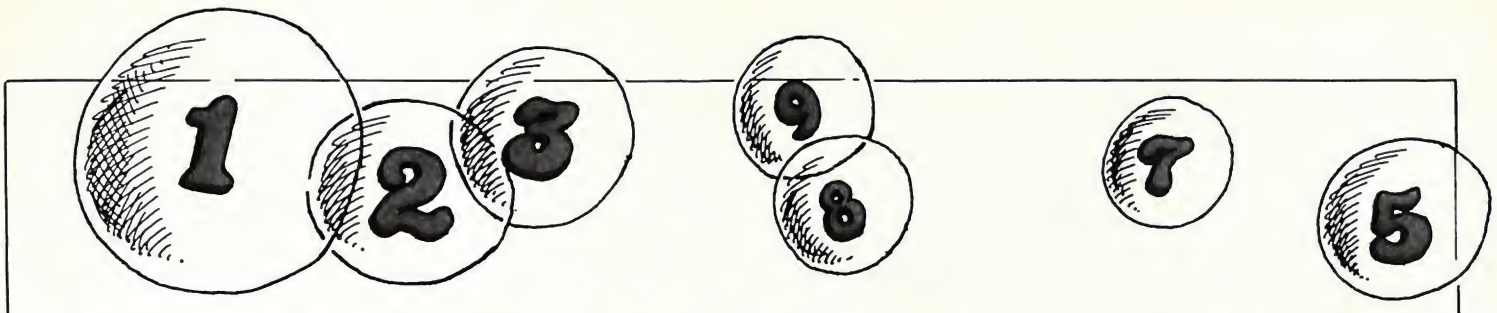
The loop is repeated until no exchanges are made and J=0=B. When B=0, line 210 will transfer control to the next segment of the program—in this case, the printout.

```
320 REM....PRINT....
330 FOR X = 1 TO N
340 PRINT A$(X);"  " ;
350 NEXT X
360 END
```

Finally, here is the long-lost DATA statement. As a handy "Benchmark" for comparing sorts, I often use the sequence found on the typewriter keyboard:

```
370 DATA Q,W,E,R,T,Y,U,I,O,P,A,S
D,F,G,H,J,K,L,Z,X,C,V,B,N,M,END
```

Finally . . . now's the time . . . run it, and count the seconds it takes to sort (about 8 seconds on an Apple II Plus).

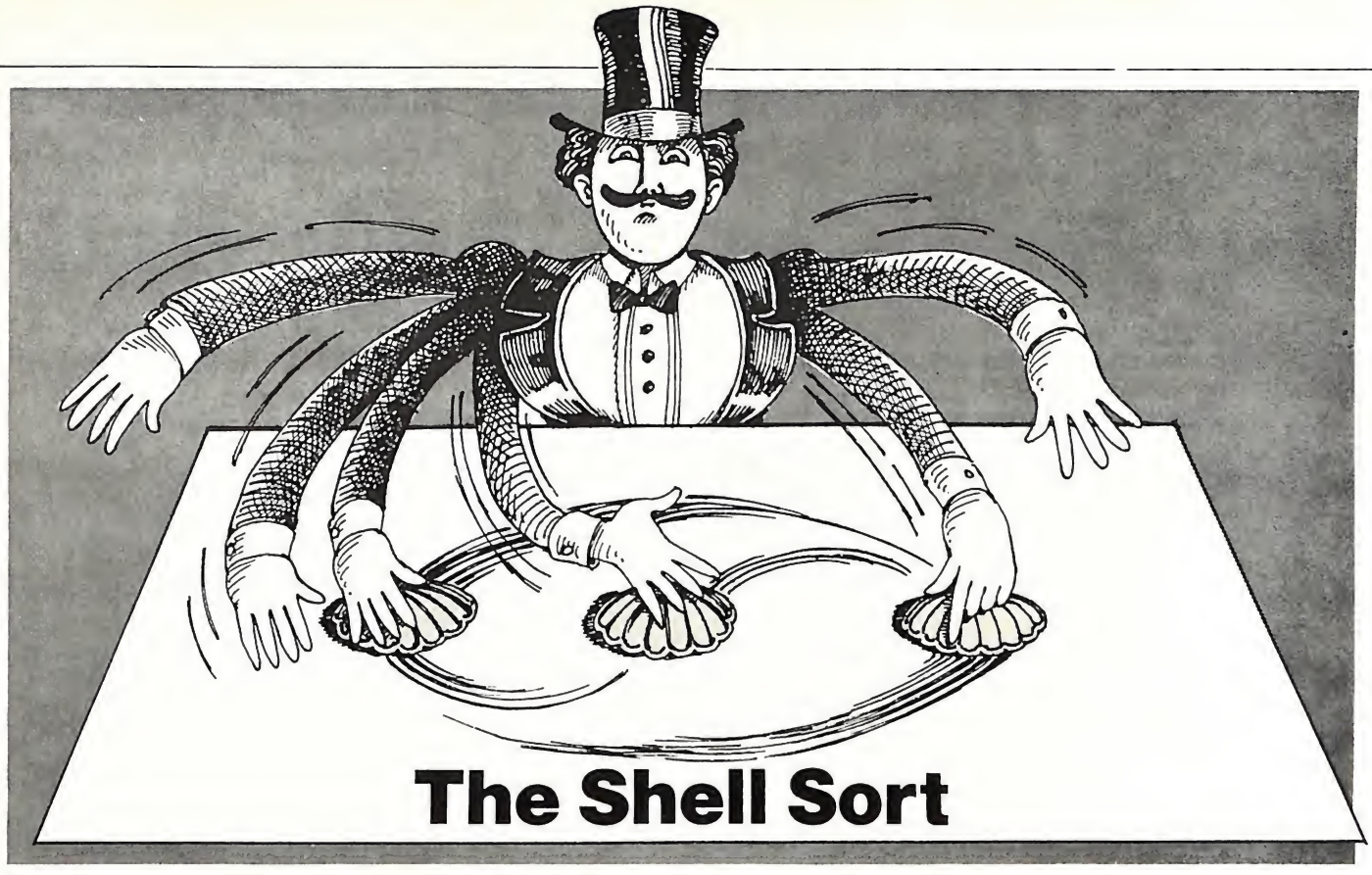


For kicks, change the DATA line to alphabetical order:

```
370 DATA A,B,C,D,E,F,G,H,I,J,K,L
    M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
```

Now run it again . . . surprise? It prints almost immediately! So our "Boundary Bubble Sort" does work, and it is an improvement over a plain bubble sort.

```
100 REM  >> PGM: BBSORT <<
110 DIM A$(100)
120 REM ....LOAD ARRAY....
130 N=1
140 READ A$(N)
150 IF A$(N)="END" THEN 180
160 N=N+1
170 GOTO 140
180 N=N-1
190 REM ....SORT....
200 B=N
210 IF B=0 THEN 330
220 J=0
230 FOR I=1 TO B-1
240 IF A$(I) <= A$(I+1) THEN 290
249 REM ....EXCHANGE....
250 T$ = A$(I)
260 A$(I) = A$(I+1)
270 A$(I+1) = T$
280 J=I
290 NEXT I
300 B=J
310 GOTO 210
320 REM ....PRINT....
330 FOR X=1 TO N
340 PRINT A$(X);" ";
350 NEXT X
360 END
370 DATA Q,W,E,R,T,Y,U,I,O,P,A,S,
    D,F,G,H,J,K,L,Z,X,C,V,B,N,M,
    END
READY.
```



The Shell Sort

The computer world is indebted to Mr. D.L. Shell for this sort which appropriately bears his name. I don't know what language he used initially but using his theory I came up with this BASIC version. It should perform as prescribed on practically any computer.

While essentially another bubble sort, the Shell sort improves its efficiency by making some radical moves early in the game. Instead of comparing adjacent items in the list (as with a standard bubble sort), this sort compares items across a gap. This gap (call it G) starts out at 1/2 the length of the list. Therefore, in the first pass, members of the list can be moved from the middle all the way up to the top, or, from the end, up to the middle. This provides a very fast "rough sort" and accounts for the overall greater speed.

After all the exchanges are made that can be with G equal to 1/2 the length, the gap is halved again, and so on until the gap ends up at 1. The final passes (with a gap of 1) are actually a regular bubble sort.

Our demo program will use a data array for testing, so let's jump right in . . .

```
100 REM  --> PGM: BBSORT <--
110 DIM A$(100)(Dimension the array)
120 REM.....LOAD ARRAY.....
130 N = 1          (Set the counter)
140 READ A$(N)     Read the data
150 IF A$(N)="END" THEN 180
(Check for end of data)
160 N = N + 1      (Increment the counter)
170 GOTO 140       (Go back for next data item)
180 N = N - 1      (Adjust array to exclude
"end" item)
```

Now the data has been read into the array with N items and we're ready to get into the actual sort. As we said, we're calling the gap G and to start it is set to 1/2 the total length of the list. In our test case, there are 26 items in the list so N = 26 and G = 13.

```
190 REM.....SORT.....
200 G = INT(N/2)
```

The integer function is used to clip all the fraction if N is an odd number. It's always a good idea to check right away to be sure that there are actually items in the list. Also, this line will be used later to signal the end of the sort.

```
210 IF G = 0 THEN 350
```

Time to step through the array list:

```
220 FOR I = 1 TO N-G
```

I will vary from 1 to (26 – 13) = 13. It is in the comparisons coming up that the Shell sort takes on its personality. The first item [A$(I)] is compared—*not* with the adjacent item, A$(I + 1)—but with A$(I + *G*) where *G* is the gap. In this case, I + G = 14. So, if the first item is less than or equal to the 14th item, nothing is done. But if it is greater, the two are exchanged:

```
230 IF A$(I) <= A$(I+G) THEN 290
240 REM.....EXCHANGE.....
250 T$ = A$(I)
260 A$(I) = A$(I+G)
270 A$(I+G) = T$
```

We must have some way to know when the list is ordered as well as it can be for each gap setting, so we establish a flag E which records and exchanges. When an exchange is made, E is set to the value 1.

```
280 E = 1
```

Then I is incremented and the next comparison is made using the same gap setting.

```
290 NEXT I
```

What happens during the comparisons in the first complete pass is illustrated in Figure 1. The comparisons

In our list, *Q* is the first item and *F* is the 14th and they are exchanged. The standard three-step routine is used

with the temporary variable T$:
which result in an exchange are indicated with an "x". You can see that some items have really moved up the list: *F, G, J, K,* and *L* are near the top while *Q, W, R, T,* and *Y* have been moved over halfway down the list. These are pretty good moves for just one pass and are proof that the "gap" comparisons accomplish their purpose. To see if exchanges have been made we check the flag:

```
300 IF E = 0 THEN 330
```

If exchanges *have* been made, E will equal 1 (line 280) and we must reset the flag to 0 so that we can check again after the next pass:

```
310 E = 0
```

Then back for another pass (using the same gap):

```
320 GOTO 210
```

If exchanges *have not* been made during the pass, E will equal 0 which will be detected in line 300. Control will pass to line 330 where we will cut the size of the gap in half. Once again, the integer function will chop off the remainder, if any: *(see footnote)*

```
330 G = INT(G/2)
```

And so back up to make another pass, but this time with the gap cut from 13 to 6:

```
340 GOTO 210
```

The passes will continue with a gap of six until no more exchanges can be made. The gap will then be cut to 3; more passes; then to 1 and a clean-up as a regular bubble sort.

When no more passes are made, E will equal zero, G will equal INT(1/2) which is zero and line 210 will pass the program to the output print routine.

```
350 REM.....PRINT.....
360 FOR X = 1 TO N
370 PRINT A$(X);"  " ;
380 NEXT X
390 END
```

Of course, we must have data:

```
400 DATA Q,W,E,R,T,Y,U,I,O,P,A,
    S,D,F,G,H,J,K,L,Z,X,C,V,B,N
    M,END
```

So you see that the Shell sort is not very difficult to code. To prove that it is worthwhile to use, try some comparison runs with other sorts and other benchmark data. The added speech can be signficant in long lists where every little bit helps. And that's our R_X for disorders. C=

Footnote:
For you Theoreticians: If the integer function were not used in line 330, the program would go into an infinite loop when the gap became less than 1. G=(G/2) would never reach zero and line 210 would never be true and the program would never stop. I know—because I fell into that trap on the very first trial run of this program!

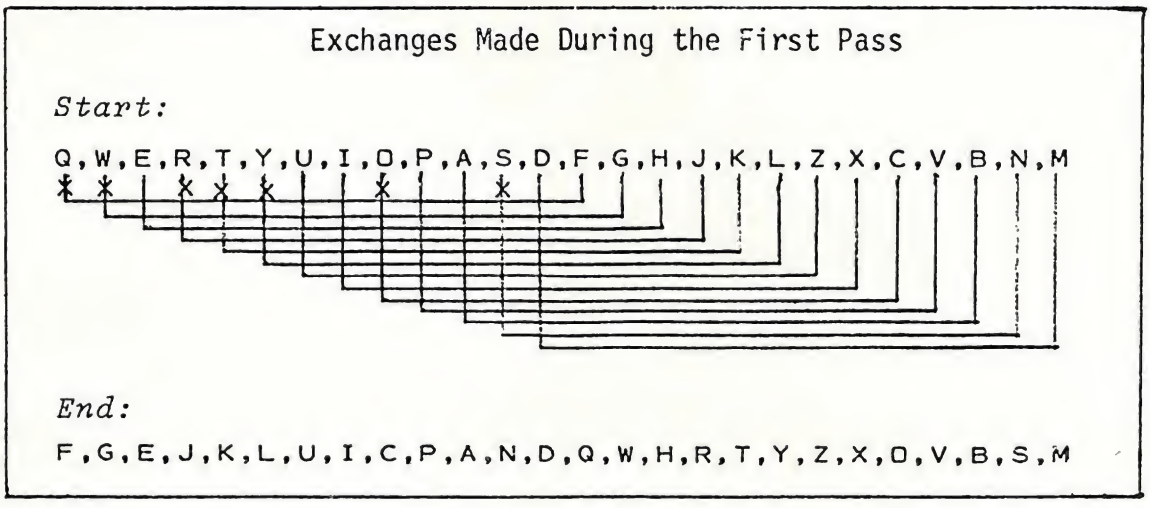


Figure 1.

```
100 REM   >> PGM: SHELL SORT <<
110 DIM A$(100)
120 REM ....LOAD ARRAY....
130 N=1
140 READ A$(N)
150 IF A$(N)= "END" THEN 180
160 N=N+1
170 GOTO 140
180 N=N-1
190 REM ....SORT....
200 G=N/2
210 IF G=0 THEN 350
220 FOR I=1 TO N-G
230 IF A$(I) <= A$(I+G) THEN 290
240 REM ....EXCHANGE....
250 T$ = A$(I)
260 A$(I) = A$(I+G)
270 A$(I+G) = T$
280 E=1
290 NEXT I
300 IF E=0 THEN 330
310 E=0
320 GOTO 210
330 G=INT(G/2)
340 GOTO 210
350 REM ....PRINT....
360 FOR X=1 TO N
370 PRINT A$(X);" ";
380 NEXT X
390 END
400 DATA Q,W,E,R,T,Y,U,I,O,P,A,S,D,F,G,H,J,K,L,Z,X,C,V,B,N,M,END
```

Joysticks Revealed

by Steven Hall

Having had several enquiries on how to use a joystick in your own programs, I feel it would be of interest to VIC-20 users to include the following.

The main function of the control port, which the joystick is plugged into, is to received variable values from the pin connections. To access these variable values one must PEEK the relevant registers. Peeking is, as the name implies, looking into the computer's memory banks and seeing what value is there.

The correct registers to look into, are 37151 and 37152. Now, if the value in 37151 is 122, then this means the joystick has been pressed upwards (AWAY), a value of 110 means LEFT, 118 means down and finally 94 means you are pressing the fire button! Easy isn't it! Unfortunately – if you are checking for right, you must stop the keyboard scan for a second by poking 37154 with PEEK(37154) AND 127 and then check 37152 for a value of 128.

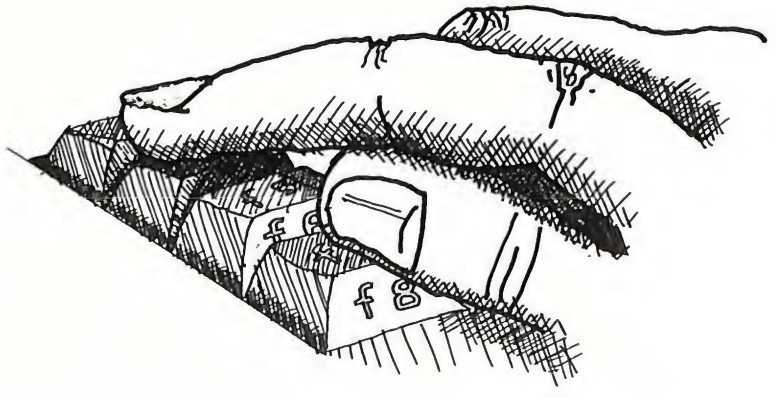
This whole operation is neatly performed with the following program.

```
10  A=PEEK(37151)
15  GOSUB 100
20  IF A= 126 AND B=247THEN 10
30  IF(A AND 4)=0 THEN PRINT"TOP";
40  IF (A AND 8)=0 THEN PRINT
    "DOWN";
50  IF (A AND 16)=0 THEN PRINT
    "LEFT";
60  IF (A AND 32)=0 THEN PRINT
    "BANG";
70  IF (B AND 128)=0 THEN PRINT
    "RIGHT";
99  PRINT :GOTO 10
100  POKE 37154, PEEK(37154)
    AND 127
120  B = PEEK(37152)
140  POKE 37154,255
160  RETURN
```

This leads to a great avenue of projects, for example, creating tunes by using the values from the joystick to be input to the VIC-20's speakers. Or alternatively, use the output from the joystick – if you have a super expander cartridge – to draw pictures on the screen – that can be very interesting. Hint! use the fire button to clear the screen or else you will fill up with all squiggles very soon!!

So That's How You Use The P.F. Keys

by Steven Hall



The P.F. Keys (Programmable Function Keys) on the right hand side of the VIC-20, can be used within a BASIC program using the following short program. Lines 20, 100, 110 are there to check if you have the shift key held down, for example using function keys 2,4,6 or 8.

I have found the best use of the keys is with musical programs to place different values to my variables for example, if A = 44 then D = 8, and use D to change volume in a piano program etc. An increasing number of Commodore cartridges use the P.F. keys, the Super Expander letting you set up the keys with a simple KEY command. One of the new games being released shortly, Omega Race uses the keys to select which character and screen colours you want to play the game with.

The more you learn about BASIC or machine code programming the more useful those keys will become. Nowdays, I load in a short assembler routine, which automatically checks to see if I'm pressing any of the keys and then execute any commands I have set for them – very useful!

```
10  A = PEEK (203)
20  B = PEEK (653)
30  C = 0
40  IF A = 39 THEN D = 1 : GOTO 100
50  IF A = 47 THEN D = 3 : GOTO 100
60  IF A = 55 THEN D = 5 : GOTO 100
70  IF A = 63 THEN D = 7 : GOTO 100
80  GOTO 10
100  IF B > 1 THEN B = 0
110  K = K + B
120  PRINT 'YOU PRESSED
     FUNCTION KEY' ; D
130  GOTO 10
```

Here is a little tidbit that you may find interesting. Did you ever wish that you could 'hide' parts of a program, while leaving it all in BASIC and copyable? Well here is a simple and effective way to do just that!

Simply load your program into memory, and place five colons (:::::) in front of every statement that you wish to hide, after the line number and before the BASIC line itself. Then just key in the following statements and execute them via a 'goto.' List the program and you will see that your secret code has vanished! To make the lines reappear, just 'goto' the line number you assigned to the second half of the statements below.

Try this little trick on the small program below just to see it work, it's fun and only costs you a little more space for the five colons and the extra statements.

```
10 rem "simple program"
20 print"this line will not be protected"
30 print"this line will not be protected either"
40 :::::print"but this line will. . . ."
50 :::::print"and this one too. . . ."
55 stop: rem "assume that this is the logical end of the program
60 rem "the coding from 60000-60020 will make the lines vanish
70 rem "the coding from 60030-60050 will make the lines reappear
60000 :::::fori = 1024 to 15000
60010 :::::ifpeek(i) = 58andpeek(i + 2) = thenpokei,0:i = i + 5
60020 :::::next:stop
60030 :::::fori = 1024 to 15000
60040 :::::ifpeek(i) = 0andpeek(i + 1) = 58andpeek(i + 2) = 58thenpokei,58:i = 1 + 5
60050 :::::next:stop
```

PLEASE NOTE . . . the value of 15000 in lines 60000 and 60030 just needs to be any value large enough so that it is larger than the number of bytes of your code to ensure that the search for the five colons continues through your entire program.

Have any interesting little programs, tricks, etc. for Commodore micros? Let us know and we'll feature them in Commodore Magazine. Send your suggestions to:

"DID YOU KNOW"
Commodore Magazine
PO Box 336
Artarmon 2064
Australia

PETkit

Computer Cleaning Kit

PETkit

Computer Cleaning Kit

PETkit is an excellent, simple-to-use kit for all the cleaning and maintenance requirements of Commodore PET Computers, Floppy Disc Drives and Cassette Decks.

Periodic inspection and cleaning of R/W heads of disc drives and cassette decks will ensure error-free data capture and transmission.

Contents

1	Aerosol SAFECLENE
1	Aerosol FOAMCLENE
100	SAFEBUDS
10	SAFECLOTHS
100	SAFEWIPES (10 cm square)
10	SAFECLENS Sachets
1	FLOPPICLENE 5¼" Jacket
10	FLOPPICLENE Cleaning Discs
1	Glove

PET, Floppy Disc Drive and Cassette Deck Maintenance
Spray FOAMCLENE liberally on to external surfaces and wipe off with a SAFECLOTH. Repeat as necessary. Avoid spraying FOAMCLENE directly on to keyboards which should be cleaned with SAFEBUDS moistened with FOAMCLENE only.

Floppy Disc Drive Maintenance
Read instructions carefully on the FLOPPICLENE jacket. To clean the heads of Drive O, type the following lines:

```
10 OPEN 1, 8, 15
20 FORA = 1 TO 5: PRINT # 1, "I0": NEXT
30 CLOSE 1
```

Insert the prepared FLOPPICLENE into Drive O with the label face up, then type RUN. The heads will be cleaned and the screen will display READY.

To clean the heads of Drive 1, type the following lines:

```
10 OPEN 1, 8, 15
20 FORA = 1 TO 5: PRINT # 1, "I1": NEXT
30 CLOSE 1
```

Insert the prepared FLOPPICLENE into Drive 1 with the label face up, then type RUN. The heads will be cleaned and the screen will display READY.

NB: Type in the lines exactly as shown, without any extra spaces.

AVAILABLE NOW FROM YOUR COMMODORE DEALER

Alternatively, if you are in a hurry and want to clean the R/W head of an 8050, you may insert the prepared FLOPPICLENE in the drive, press the door latch and wait until the drive stops. Release the door latch and repeat the process five or six times.

Load pads should be examined periodically and replaced as required. No attempt should be made to clean these pads.

NB It is good practice to wear the glove supplied when carrying out cleaning operations.

Cassette Deck Maintenance
Press eject/stop button to lift tape housing lid. Press play button to allow easy access to R/W and erase heads. Take a SAFEBUD and soak it with SAFECLENE spray. Clean the heads from the top with an up and down motion of the SAFEBUD.

NB DISCONNECT THE CASSETTE DECK FROM THE PET BEFORE CLEANING

VDU Screen
Apply a SAFECLENS Anti-Static Screen Wipe to the VDU screen, then wipe the screen until it is dry with a clean SAFEWIPE.

Remove foam packing strips from under aerosols prior to using this PETKit.

AF ELECTROLUBE

commodore COMPUTER

VIC CHIP MEMORY USAGE.

by Garry Mason

Because of the memory addressing restrictions of the VIC chip, it is necessary to have the screen located below $2000. RAM based character generators must also be located below $2000, because of the internal addressing restrictions of the 6561 (VIC chip).

When the 3K Super Expander is used with a memory expansion cartridge such as the 16K or 8K expansion, the 3K Super Expander RAM will be mapped out of the BASIC memory map. This remapping is to ensure continuous memory from $1200 for BASIC programs. The 3K of RAM in the cartridge can only be accessed using PEEK and POKE or assembly language routines. Using a 3K cartridge in the same configuration will have the same effect.

The VIC chip can use the ROM based character generators or a RAM based character generator by loading the VIC register at 36869 with the appropriate value indicated by the X in the following table. RAM based character generators should be protected against corruption by BASIC programs by resetting the start of BASIC or End of BASIC pointers in zero page.

POKE 36869, PEEK (36869) AND 15 OR (X*16)

X Value	Location HEX	DECIMAL	Contents
0	8000	32768	Upper case normal characters
1	8400	33792	Upper case reversed characters
2	8800	34816	Lower case normal characters
3	8C00	35840	Lower case reversed characters
4	9000	36864	unavailable
5	9400	37888	unavailable
6	9800	38912	VIC chip – unavailable
7	9400	39936	ROM – unavailable
8	0000	0	unavailable
9	0400	1024	Only with 3K cartridge
10	0800	2048	Only with 3K cartridge
11	0A00	3072	Only with 3k cartridge
12	1000	4096	RAM
13	1400	5020	RAM
14	1800	6144	RAM
15	1C00	7168	RAM

Commodore 9060/9090 Hard Disk Drive continued from Page 13

SuperDOS. This DOS is also available in the CBM 8250 floppy disk drive and may become available for the CBM 8050 drive. The major enhancement of SuperDOS is that it can support very large relative files. The DOS supports a super side sector that contains track/sector pointers to 127 groups of 6 side sectors for a maximum relative file size of 23.25 Megabytes. This is much larger than any existing CBM storage peripheral and allows for expansion to larger hard disk drives should they become available.

Apart from this important feature, the disk responds to the same command structure as DOS 2.5. The execution speed of some commands has not been improved by the hard disk, as a 'HEADER' command will take 105 minutes to format a CBM 9090 disk drive.

PROGRAMMING WITH THE HARD DISK

As the disk is completely compatible with the existing range of Commodore storage peripherals, there are few problems in developing an application using the disk peripheral. The biggest problem is resolving the use of Drive 0 for program disks and Drive 1 for data disks. This can involve extensive changes to software, especially software that may still have to be able to be used in a floppy environment. The best approach is to use a system parameter file and to use parameter driven file statements.

It is possible to establish a very long directory if the disk is being used to store large numbers of files, this can make life tedious if the user is searching for a particular file. Most standard disk utilities will work with the hard disk, provided the appropriate track and sector ranges are modified.

Backup is a problem that all programmers will have to consider. It is possible to copy any file that will fit onto a CBM 8050, to an 8050 drive. As the copy operates at IEEE data transfer speed, a complete backup from hard disk to floppy diskette will take several hours. Most programmers will be able to write short programs to copy selected files onto floppy diskette. The difficult problem involves backup of very large relative files that exceed the 719 block limit of an 8050 relative file. The only option is to write a utility program that can dump a relative file by splitting the file into several 8050 size relative files, then rebuild the file when a reload of the hard disk is required. This type of utility is not difficult to write and involves manipulation of the disk BAM and side sector tables.

USING THE HARD DISK

The hard disk is a delight to use. The Commodore Accounting software has been mounted on the CBM 9090 disk at Commodore and provides a very different environment to the 8050 version. Response to any request for information that requires disk access is immediate. Existing versions of VISICALC and WORDCRAFT can be used with the hard disk.

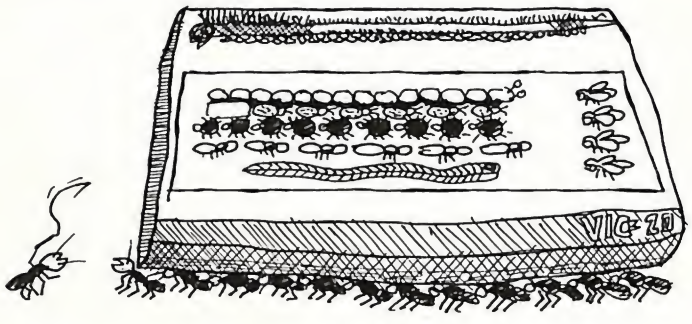
The hard disk provides a superb environment for program development as the tedium of disk swapping and continual disk accesses for programs and data is dramatically improved. The productivity of program developers usually improves dramatically if a hard disk is available.

Over the next few months we will start to see some exciting applications for CBM computers using the hard disks. If you are interested in using a hard disk, contact your Commodore Dealer for further information.

VIC 20 OPERATING SYSTEM BUGS

by Robert Baker

Everyone should be aware by now that computer systems do not always function 100% by the book. Occasionally, small errors or operating system quirks appear and we can usually live with them once documented and well known. More serious problems may require upgrade or replacement ROM's or program patches to correct problems that cannot be programmed around by the user.



Well, the VIC-20 is no exception and I've come across two problem areas so far that bear mentioning. The first problem has to do with the INPUT command and line wrapping on the VIC's 22 column display. The problem appears when you print a prompt for an input command and the prompt is longer than 22 characters, causing the display line to wrap to a new line. This is assuming that the prompt and the input variable are separated by a semicolon so that they're expected on the same line. Whenever this situation arises, the prompt itself is returned along with any keyboard entry made by the user.

When the program is expecting a string input (A$), you simply get more characters than you expected, but the program still runs. If the program is expecting a numeric response (A or A%), then you get a REDO FROM START error message. The input prompt is re-displayed, and the program waits for new input from the user. Now you have a problem, since no matter what you enter, the input prompt is returned to the program along with whatever keyboard entry is made. Since the prompt is not a valid number, the system is hung in a loop it cannot get out of.

If you press the RETURN key without entering any characters first, the system simply returns the last keyboard response entered on the VIC so you're still stuck in the loop. Even the RUN/STOP key cannot always get you out of this error condition since the program is not really executing any BASIC statements.

Here's a little test program that will quickly demonstrate what is happening in your VIC:-

```
10 INPUT"PLEASE ENTER ANY
   TEST REPSONSE";A$
20 PRINT A$
```

When you enter your response and press [RETURN], you will see the entire prompt re-displayed followed by your keyboard entry. If you now type in, PRINT A$ as an immediate command, you can prove to yourself that the A$ response the program received is really as shown with the prompt included. If you change the A$ to A or A% in both lines, you'll now see the REDO FROM START error when you try to run the program again. You'll even see the error when you input a valid number response. Now try pressing the RETURN key alone or the RUN/STOP key to get out of the loop. There are some ways to get out of the loop but if all else fails just turn off the VIC and start again. Fortunately, you can easily overcome this problem simply by adding a null string to the beginning of the PRINT# statement:-

```
20 PRINT#3,"";SPC(5);"HI THERE"
```

to make everything work correctly. You see, the SPC or TAB only causes the problem if it's the first item in the PRINT# statement, any other place is OK. In this example, I used device #3, the display screen, since it's the simplest device to run test cases on. The problem does, however, occur on all devices using the PRINT# command, including the printer!

These errors have been reported, but there has been no response as to how or when they'll be fixed. As long as we know about them, they can be avoided.

By the way, this same problem occurs if you use a PRINT command for the prompt and an INPUT command to get the response, if there is a semicolon at the end of the PRINT argument. Try this little example:-

```
10 PRINT"PLEASE ENTER ANY
                TEST RESPONSE";
20 INPUT A$
30 PRINT A$
```

You should see the same results as before. This problem is annoying but can be avoided. The best methods are to either limit input prompts to a single line or get the input on a new line without using the semicolon in the PRINT statement.

As I mentioned earlier, there is another problem area in the VIC-20 besides the one just described. The second problem involves the PRINT# command. When you open a logical device and use the PRINT# command to output to that device, you'll get a SYNTAX error if the first argument of the PRINT# statement is a TAB(..) or SPC(..) function. This little example shows what happens on the VIC:-

```
10 OPEN 3,3
20 PRINT#3,SPC(5);"HI THERE"
```

You should see a SYNTAX error reported on line 20. You'll get the same result if you replace SPC with TAB. The line is perfectly correct according to the manuals and will execute.

KEYBOARD DECODING

by Garry Mason

The following is a list of all the values that can be produced via the keyboard matrix variable, which can be found in zero page. This variable is produced by a routine which reads the keyboard X and Y scanning co-ordinates to produce the following values. This value corresponds to an offset. This when added to a base address, gives the position of information in ROM or RAM, concerning that character. In other words it is used as a pointer for use in a look-up table. Once this is done, the routine combines this with other information, such as 'is the SHIFT key pressed?', to produce the actual character value.

Some programmers use this variable in lieu of the GET. The advantage being that this variable repeats, where as the the GET statement would not. For example, in a games situation the player holds down the '4' key expecting the program to continually pick the '4' up and act accordingly. If the program uses the GET statement it will not pick up any more than one character. Whereas a program that uses the keyboard matrix co-ordinates variable would continue to pick up the '4' key and then act.

This table will help when converting programs from one Commodore Computer to another. The location which was PEEKed for all new ROM BASIC 4.0 machines (4032, 8032 and 3032) was 151 – matrix co-ordinates of key down. This converts to location 197 on the VIC, being a BASIC 2.0 machine and location 515 on old ROM BASIC 1.0 machines (2000 Series).

As you may have noticed, all the characters available on the keyboard are not represented here, this is because the use of the shift key or other control key have no effect on the value returned, these codes only refer to the physical position on the keyboard of each key. For example there is no difference between the code returned from the cursor down key and the code returned from the cursor up key. So in the table they are listed as 'vertical cursor' key.

NOTE: The 8032 as you may know has two number 'pads' on the keyboard. For this reason the second set (the ones not found on the number pad) are shown as the punctuation marks. The punctuation section is missing from the VIC keyboard – this is because all punctuation marks on the VIC keyboard are supplied as shifted numbers.

CHARACTER CODES

CHARACTER CODES	VIC:	3032, 2001 Series 4032	FAT 4032:	8032:
left arrow	8	75	95	223
1	0	26	49	177
2	56	18	50	178
3	1	25	51	179
4	57	42	52	180
5	2	34	53	181
6	58	41	54	182
7	3	58	55	183
8	59	50	56	184
9	4	57	57	185
0	60	10	48	176
+	5	17	43	
–	61	9	45	45
HOME	62	74	19	19
DELETE	64	65	20	20
Q	48	64	81	81
W	9	56	87	87
E	49	63	69	69
R	10	55	82	82
T	50	62	84	84
Y	11	54	89	89
U	51	61	85	85
I	12	53	73	73
O	52	60	79	79
P	13	52	80	80
@	53	15	64	192

CHARACTER CODES	VIC:	3032, 2001 Series 4032	FAT 4032:	8032:
*	14	33	42	
↑	54	59	94	222
A	17	48	65	65
S	41	40	83	83
D	64	47	68	68
F	64	39	70	70
G	64	46	71	71
H	64	38	72	72
J	20	45	74	74
K	64	37	75	75
L	21	44	76	76
:	45	36	58	58
;	22	28	59	59
=	46	1	61	
Z	33	32	90	90
X	26	24	88	88
C	34	31	67	67
V	27	23	86	86
B	35	30	66	66
N	28	22	78	78
M	36	29	77	77
,	29	21	44	44
.	37	2	46	46
/	30	49	47	47
!		80	33	49
"		72	34	50
#		79	35	51
$		71	36	52
%		75	37	53
&		74	38	54
'		70	39	55
(		76	40	56
)		68	41	57
+		7	91	219
←		14	93	221
vertical cursor	31	66	36	17
horizontal cursor	23	73	75	29
English Pound	6			
F7	63			
F1	39			
F5	55			
F3	47			

USING THE VIC 1515 PRINTER

by Garry Mason

The VIC printer is probably one of the most intelligent devices that you may have around the home, except the VIC of course.

I mean, how would you like to be in its position; not only does it have to control the data flow on the serial bus but it also has to control a lot of hardware as well. What with the head-movement, the paper feed – both of which have to be perfectly controlled to get the one hammer to hit the barrel as it flies past at exactly the right time.

This is the one printer that employs characteristics not seen on most CBM printers, the extensive use of control codes. This means we do not have to even consider the use of secondary addresses.

There are in fact two of these secondary addresses, but as they can be accessed by control codes, we need not worry about them.

Control codes are in fact characters which, when they get to the printer, are interpreted as something other then the usual everyday character. When these control characters reach the printer, they are translated into actions which change the way in which the printer responds to data.

This is very much like the way the VIC can put control characters in print statements to produce special effects on the screen. For example if the CONTROL and the RVS key are used in a print statement, the VIC will print all following characters in reverse field until told to stop. This can be duplicated on the printer, just by including the same control character in the PRINT# statement.

```
10  A$ = "[RVS]THIS IS IN REVERSE
    FIELD[OFF RVS]
20  OPEN4,4
30  PRINT A$ :REM PRINT ON
    SCREEN
40  PRINT#4, A$
    :REM PRINT TEXT ON
50  CLOSE 4 : END
```

The control codes in this example were RVS and OFF/RVS, these two control codes can be inserted into text by using the VIC screen editor. Other control codes cannot be sent to the printer in this manner and require the user to use this type of approach.

```
10  OPEN 4, 4
20  PRINT#4, "CHR$(18);"THIS IS

IN REVERSE FIELD";CHR$(146)
30  CLOSE 4 : END
```

In this example we have inserted the control code which means turn on reverse field printing in the form of CHR$(18). The character code (18) means exactly the same as the RVS symbol, as it is the number 18 that gets sent to the printer anyway.

UPPER & LOWER CASE

When the VIC-20 is turned on it is set in upper case mode. This means that keys typed without the shift key will turn out to be in upper case and keys typed with the shift key will come out as graphics. This is the same as the printer when it is also powered on. If, on the other hand, you want your printer to act like the VIC when in lower case mode, you must use the secondary address which does this. This will not affect the printer in any other way.

```
OPEN 4, 4, 7    instead of    OPEN 4, 4
```

LINE SPACING

The next thing that can be easily set is the line spacing. This controls how far apart the lines are from each other. Normally there are 6 lines to the inch. If the printer is in graphics mode, i.e. CHR$(8) has been sent at some time, then the printer will squeeze 9 lines to the inch. This has the effect of having no space between the lines so a vertical line will be printed without any gaps down the page. This feature can be demonstrated as follows:-

```
10  OPEN 4, 4
20  FOR C=1 TO 5
30  PRINT #4, CHR $(15);"! THIS IS A
    VERTICAL LINE";CHR$(8)
40  NEXT C
50  CLOSE 1 : END
```

Note that the print line only turns on graphics mode to do the linefeed, and then turns on text mode straight away. This is because no text can be printed in graphics mode.

COLUMNS & HEAD POSITIONING

Columnising fields is usually done on most printers by using the TAB or PRINT USING statement. This is not really possible on the VIC printer, especially because the PRINT USING statement is not supported in Commodore BASIC.

The TAB statement is a relative implimentation; this means that if a tab is given, it will be added to the present print head position. Whereas it should calculate all printing positions from the left margin.

To get around this little problem, programmers usually send the printer carriage back to the left margin (without moving the paper up) and TABbing out to the next column. Unfortunately the VIC printer does not have this facility.

All is not lost though, as the VIC printer provides its own form of TAB. This is done via the control code POS or CHR$(16). This can be used in two ways, depending upon which mode is required. Firstly you can specify the column number to start printing on, or secondly you can specify which dot on the printer you want printing to begin on.

```
PRINT#1,CHR$(16);"10 HI THERE";
    CHR$(16);"40 HI THERE"
```

The above example uses the control mode POS to position the character positions. Only 80 positions or columns can be TABbed to, using this method.

Using the dot positioning command is not as simple as it first looks. The first thing to consider is how many positions there are on a line; well the VIC printer prints each character using a 5 by 7 matrix. Seven vertical dots by 5 horizontal dots. So to position the print head to be in line with the normal character position we have to move the print head in sets of six dots – five character positions plus one for spacing. This type of addressing is done via the same control code (POS) but with the ESC or escape code prefixed to the POS code. The command then becomes:-

ESC	POS	HP	LP
(27)	(16)	(xx)	(xx)

HP and LP refer to the high and low bytes of the dot position respectively. This address is given in two bytes or bits of information. LP allows the user to access up to 255 positions, the remainder of the 480 positions are accessable by using HP. The valid values for HP are 0 (for positions 0-255) or 1 (for positions 256-480). If you wish to address the 256th position, the complete command would be (27),(16),(1),(0) and the 257th position would be (27),(16),(1),(1).

The algorithum for calculating the values where DA is the dot address is;

HP= INT(DA/256) LP= DA-HI*256

The following program shows how this can be used to position text, first by character position, then by dot address.

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
                                        COLUMN40                COLUMN60

                                     VIC 1515 PRINTER
                                     VIC 1515 PRINTER
                                     VIC 1515 PRINTER
                                     VIC 1515 PRINTER
                                     VIC 1515 PRINTER
                                     VIC 1515 PRINTER
                                     VIC 1515 PRINTER
                                     VIC 1515 PRINTER
                                     VIC 1515 PRINTER
                                     VIC 1515 PRINTER
                                     VIC 1515 PRINTER

READY.
```

```
10 OPEN4,4
20 REM PRINT COLUMN NUMBER
30 A$="1234567890"
40 FOR C=1 TO 8
50 REM PRINT#4,A$;
60 NEXTC
70 REM PRINT#4,
80 REM USE CHARACTER POSITION ADDRESSING
90 REMPRINT#4,CHR$(16);"40COLUMN40";CHR$(16);"60COLUMN60"
100 REM PRINT#4,
110 REM USE DOT POSITION ADDRESSING
120 FOR C=250TO260
130 LP=C:HP=0:IFC>255THEN HP=1:LP=LP-256
140 PRINT#4,CHR$(27);CHR$(16);CHR$(HP);CHR$(LP);"VIC 1515 PRINTER"
150 NEXTC
160 CLOSE4:END

READY.
```

Program 1

The short program below uses this feature to spread out the characters in a line to text equally, thus producing an even right margin. It will work correctly as long as the inputted string does not exceed eighty characters.

This program is only a sample to demonstrate the attractive output which may be produced. The main routine is in lines 6 to 30 and may be incorporated into other programs, such as a word processor, to produce very nice results.

```
5  INPUT A$: OPEN 4,4: CMD4
6  IF RIGHT$(A$,1)=" " THEN A$=
   LEFT$(A$,LEN(A$)-1): GOTO 6
10  A=LEN(A$): B=(474/(A-1))
20  FOR C=0 TO A-1: D=C*B: D=INT
   (D/256): E=C*B-D*256
30  PRINT CHR$(15);CHR$(27);
   CHR$(16);CHR$(D);CHR$(E);
   MID$(A$,C+1,1);
35  NEXT C
40  PRINT#4: CLOSE 4
```

CHARACTER DEFINITION

Defining your own characters is also possible by using graphics mode. In text mode every character or byte of information sent to the printer is interpreted as one character that occupies one character matrix.

In graphics mode, each byte is interpreted as 7 dots in the vertical plane of the current head position. If text is sent to the printer in this mode only garbage will be printed – if anything at all.

Graphics mode is entered by using the control code BS or CHR$(8) and is exited by using CHR$(15) or S1. Remember that the use of these commands also affects the line spacing as already shown. If 6 lines per inch are required with graphics mode then the user will have to switch back to text mode before sending the RETURN character at the end of each PRINT# statement.

Getting back to defining characters; the way in which lines are defined is quite straight-forward. Each byte or piece of information represents one vertical line on the printer, each vertical line being made up of 7 dots.

Each of the seven dots can be given a value. When the total value for each vertical is added up as follows; we are left with a value (less than 255) that represents that pattern. The only abnormality with this is that you are required to add on 128 to each of these values.

VALUE:	DOT:		
1	0	0	
2	0	0	
4	0	*	4
8	0	0	
16	0	*	16
32	0	0	
64	0	0	
+128	+128		128+
TOTAL	128		148

from this we can form a whole string of these bytes. If we make up 6 of these vertical lines then we have enough data to forge a character.

Taking the following example, we end up with the following pieces of data:-

0	0	*	*	0	0
0	*	0	0	*	*
*	0	0	0	*	0
*	0	0	0	0	0
*	0	0	0	*	0
0	*	0	0	*	*
0	0	*	*	0	0
TOTAL					
156	162	193	193	182	162

To complete this we need to insert two control characters on either side of the data. One in front, to switch the printer in graphics mode and one behind, to put the printer back into text mode. **See Program 2.**

```
THE C PRINTER GIVES YOU VALUE FOR MONEY, AS DOES THE C DISK UNIT.


10 OPEN4,4
20 REM READ IN CONTROL CHARACTERS
30 B$="" :READ N :B$="" :FORC=1 TO N :READ B :B$=B$+CHR$(B) :NEXT C
40 REM ADD ON CONTROLS
50 B$=CHR$(8)+B$+CHR$(15)
60 REM PRINT TEXT
70 PRINT#4,"THE ";B$;" PRINTER GIVES YOU VALUE FOR MONEY, AS DOES THE ";B$;"
DISK UNIT."
80 REM COUNT
90 DATA 6
100 REM GRAPHICS DATA
110 DATA 156,162,193,193,182,162

READY.
```

Program 2

This type of program could be used to print whole lines of dots on the printer, if enough data was collected. In fact you can even do screen dumps of the VIC on a dot by dot basis.

Another on of the features would be the repeat graphics function. This function is very simple in operation, what it does is to repeat one piece of data that is sent after the repetition number. This can be used for printing a pattern. The syntax for repeating graphics data is:-

SUB rep# data

26 (nn) (xx)

This will only work if the printer is already in graphics mode when it receives the SUB code. If more than one piece of data is given, only the first piece will be repeated until the end of the repetition when the rest will be printed. The following program shows this:- **See Program 3.**

```
                ◇ END OF GRAPHIC MODE
                ——◇ END OF GRAPHIC MODE
                ————◇ END OF GRAPHIC MODE
                ——————◇ END OF GRAPHIC MODE
                ————————◇ END OF GRAPHIC MODE


READY.


10 OPEN4,4
20 REM READ IN CONTROL CHARACTERS
30 A$="" :FORC=1TO6 :READ A :A$=A$+CHR$(A) :NEXT C
40 REM READ IN REPEAT STRING
50 READ N :B$="" :FORC=1 TO N :READ B :B$=B$+CHR$(B) :NEXT C
60 REM DO LOOP CHANGING NUMBER OF REPEATS
70 FOR C=1 TO 100 STEP 20
80 R$=CHR$(C)
90 PRINT#4,A$;R$;B$;CHR$(15);" END OF GRAPHIC MODE"
100 NEXT C
110 REM CONTROL DATA
120 DATA 8,27,16,0,100,26
130 REM REPEAT COUNT
140 DATA 4
150 REM REPEAT DATA
160 DATA 136,148,162,148

READY.
```

Program 3

MAKE THE MOST OF YOUR COMMODORE

HIGH-RESOLUTION GRAPHICS

Now you can give your PET/CBM a High-Resolution Graphics capability with the MTU Graphics Hardware and Software Package. The Hardware is easily installed and the new graphics board provides five EXTRA ROM Sockets and 8k RAM MEMORY EXPANSION which can be used for program or data storage when graphics are not required. A powerful graphics Software Package is included and contains many extra BASIC commands for drawing lines defining shapes etc. The Graphics Hardware does not affect normal operation of the Commodore.

Available on all PET/CBM - BASIC 1, 2, 3, or 4.

RS232 TEST SET

The RS232 TESTSET is a small hand held device that connects inline with the interface cable, the terminal or the modem and monitors the line signals. The TESTSET passes all 25 lines through and so can be left connected without affecting communications. It is completely portable as no batteries are required since power us derived from the interface signals. Each indicator circuit is current limited to meet the requirements of the RS232 Interface Standard. Also the voltage range for activating the bright LED display corresponds with this standard and thereby reduces troubleshooting to a "GO-NOGO" problem instead of trying to measure active signals to determine voltage levels. One 2-pin and one 3-pin jumper are included.

PRINTOUT

Don't forget your PRINTOUT Magazine Subscription - for Commodore PET/CBM Lovers 12 issues p.a. from Jan. '82 incl. postage - $50.00 p.a.

VIC COMPUTING

For all VIC 20 owners, this sister to PRINTOUT is a must. 6 issues p.a. incl. postage from Jan. '82 - $25.00 p.a.

PROGRAMMERS TOOLKIT ROMS

These ROMs plug into spare sockets in your PET/CBMs and give the user additional commands such as TRACE, single STEP, FIND, RE-NUMBER, AUTO line numbering,DUMP variable contents, APPEND, and DELETE multiple lines. Also available are the DISK-O-PRO Tool-kits which gives all the extra DOS 2.0 commands to DOS 1.0 users as well as commands like PRINT USING, SCROLL and disk program MERGE -25 commands in all. The COMMAND-O provides DISK-O-PRO commands for BASIC 4.0 users.

The Programmers Toolkit for BASIC 1.0/2.0/3.0/4.0 users.
DISK-O-PRO Toolkit for BASIC 2.0/3.0 users.
COMMAND-O Toolkit for BASIC 4.0 users.

Tel: (03) 614 1433
614-1551
Telex: AA 30333.

MICROCOMP
MICROCOMPUTER SYSTEMS DESIGNERS

B.S. MICROCOMP PTY. LIMITED,
4th & 3rd Floors,
561 Bourke Street,
Melbourne, 3000.

The Link

THAT JOINS ALL OFFICE FUNCTIONS

Get it all together with the Silicon Office.

STORING AND RETRIEVING INFORMATION – CREATING, EDITING AND PRINTING OF TEXT – MATHEMATICAL CALCULATION – COMMUNICATING INFORMATION LONG DISTANCE.

Silicon Office is the first database management System for Commodore CBM Microcomputers whereby up to six files may be open and accessed simultaneously during a run. It is also the first system which permits intercommunication with fellow machines and user. The Silicon Office turns the CBM 8032 into a secretarial work station capable of emulating any application package the user cares to think of.

Now one program which is continuously and completely resident in the memory of the CBM is capable of performing all functions required to run a small business or office. This can mean anything from Accounting and Stock Control to Word Processing, Statistical analysis, mailing lists and information filing – all at once, if necessary. Combine filing cabinets, ledgers, typewriter and calculator in your office into one efficient unit.

The Silicon Office package comprises of three integrated elements: a sophisticated word processor, a flexible database management system and an option for inter computer communications – all in one memory resident program.

$8490

THIS BUSINESS PACKAGE IS NOW AVAILABLE AND WILL COMPRISE:

- 8023 DOT MATRIX/PSEUDO PRINTER
- COMMODORE CBM 8032 COMPUTER
- 8050 DISC DRIVE UNIT
- 64K ADD-ON MEMORY BOARD (TOTAL 96K RAM)
- THE SILICON OFFICE PROGRAM MASTER DISC
- TWO SECTIONAL A4 MANUALS

This package is available from:

Compute® CBM SYSTEMS

5 PRESIDENT AVE., CARINGBAH ☎ 525 5022